SENSORAMA

AN HTML5-BASED CROSS-PLATFORM SENSOR DATA VISUALIZER

Samrith Shankar

May 2014

Computing Science

Supervisor: Dr. Thomas Ploetz

Word Count: 10,256

ABSTRACT

Sensors are one of the most common detection objects around us in this modern world. They play a vital role in detecting, displaying, warning and controlling various settings. Sensor data are usually very plainly put out and there is not much description and visualization of them, thus making it difficult for the novice user to understand them. This app has been built to provide a graphical comparison of various sensor data that are available to use through the upcoming HTML5 cross-platform development language. The main aim is to make the reading of sensor data more graphical, colourful and easily understandable.

DECLARATION

"I declare that this document represents my own work except where stated otherwise."

Samrith Shankar

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Thomas Ploetz, for the immense support he has provided during the course of this project through his valuable feedback and massive inspiration. I would also like to thank all the people who are a part of the "Sensors Unlimited" group on Google Groups for their massive feedbacks and help in improving this app and making it according to the user's needs and demands. Finally, I would like to thank my parents for the massive support they have provided during the course of my Bachelor's degree.

TABLE OF CONTENTS

1. INTRODUCTION

Sensors are very important in our modern, computerised world. Sensors are very powerful devices that can detect various user specific instances like motion, rotation, location, light, temperature etc. They are also capable of performing tasks according to the value of the instance which they record. In the mobile age, where almost everyone has a smartphone, sensors have become far more common and a more of a necessity than a luxury. Smartphones come loaded with various sensors that can detect motion, rotation, location, light, temperature, gestures and user inputs.

These sensors are capable of returning readings and can perform tasks based on their readings, depending on what the developer instructs. The readings from these sensors can easily be accessed through the new HTML5 development platform, giving developers a broader scope for application development and better and innovative usage of sensors and their data.

Most sensors (except geolocation, light, temperature, sound and touch sensors) work on a three dimensional scope. They have 3 axes, namely the x-axis, the y-axis and the z-axis. The value of data across these axes differs and is unique to that particular axis for a particular sensor.

Below is a simple diagram explaining the three dimensional axes of a general sensor in a smartphone:

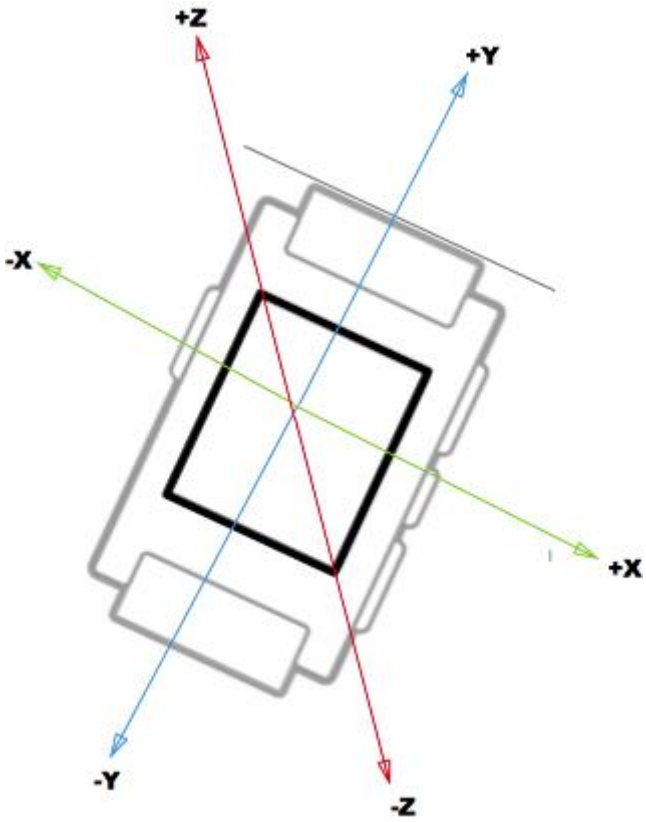


Figure 1.1 – The x, y and z-axis of a smartphone

A major problem of the sensor data is that they are usually vast and change rapidly. For a user not experienced in the field of sensor data, it sometimes becomes quite difficult to decipher the data in a simple manner. Apart from that, when users decipher data, they usually go for simplicity over precision for comparison. Though this is only a mild problem in most cases, but in some cases, loss in precision can prove to be quite costly.

This project has been developed to lessen the complexity of the sensor data and visualize it in a simple and yet very understandable manner, for comparing, calculating, researching and working with the data. The main outlook of this project is the simplicity in visualizing sensor data and the explanation of the data with respect to the real world, to give the user an idea of what the data represent and how it can be plucked around to use it for the user's needs and wants.

2. INITIAL AIMS AND OBJECTIVES

The initial aims and objectives of this project had been chosen keeping in mind the vast variety of sensors that come fitted with a smartphone. The initial idea was to make use of as many sensors as possible, using the HTML5 mobile app development framework and display their results in a simplistic and colourful manner.

2.1. HIGH LEVEL AIM

To develop a cross platform app in HTML5 for the simple visualization of sensor data.

2.2. OBJECTIVES

1. To identify the key sensors used very often in a smartphone
2. To identify how the sensor data can be visualised in a simple manner for the understanding of a novice user
3. To identify the requirements of the sensor data visualizer
4. To develop a sensor data visualizer while keeping in mind the "heaviness" of the app
5. To allow the user to record and later review the sensor data
6. To create an interactive multigraph for the sensor data across time for an effective comparison
7. To evaluate the effectiveness of the app

2.3. DESCRIPTION OF OBJECTIVES

Objective 1: A smartphone used various sensors. But in a smartphone used by the common user, only some sensors are used a lot more often, while other are seldom used. So to make this app easier for the user to relate to, a particular set of sensors had to be chosen. Those sensors had to be studied in detail and their data analysed to make the app worthwhile for the user to view.

Objective 2: Sensor data is usually displayed in a very monotonous line graph of data and with great precision. These line graphs have often been cited by user embarking on using sensor data as very difficult to decipher. Even though users were adamant on wanting graphs as it made the sensor data easier to understand, they were very much against line graphs in general. So a simple yet effective graph had to be chosen for displaying the value of the sensor.

Objective 3: Each sensor has data that is unique to that particular sensor and for each sensor, the values had to be converted into the appropriate user readable alternatives and displayed in a graph. The data visualizer had to be simple and colour code the different values for the sensor appropriately but yet display the precise values as text for users to understand the fluctuation in values.

Objective 4: The "heaviness" of the app is simple the loading time and the size of the app. With the speed provided by most smartphones, application users have little or no patience when it comes to waiting for an app to load. Hence, a lot of research had to be done in this category to minimize the loading time of the app and allow it to quickly display the data and reflect changes just as fast.

Objective 5: Recording sensor data allows the user to analyse data over a period of time and compare it with various other recordings. Since this app is cross-platform, a simple and effective solution using just HTML5 and JavaScript had to be researched to implement this particular function.

Objective 6: The multigraph is particularly useful if the user is recording data as it combines the various data, across 3 axes over a period of time, or just displays a single data over a period of time, for the user to compare and assess the sensor data. An appropriate and interactive graph had to be researched, and the feasibility of this objective.

Objective 7: The effectiveness of this app largely depended on the visualization techniques used. After the implementation of the visualization techniques, the app had to be fed in live data to test its resolve and the effectiveness. Various user tests and interface tests were conducted and the app was compared to some pre-existing applications to determine its performance.

2.4. PROJECT SCHEDULE

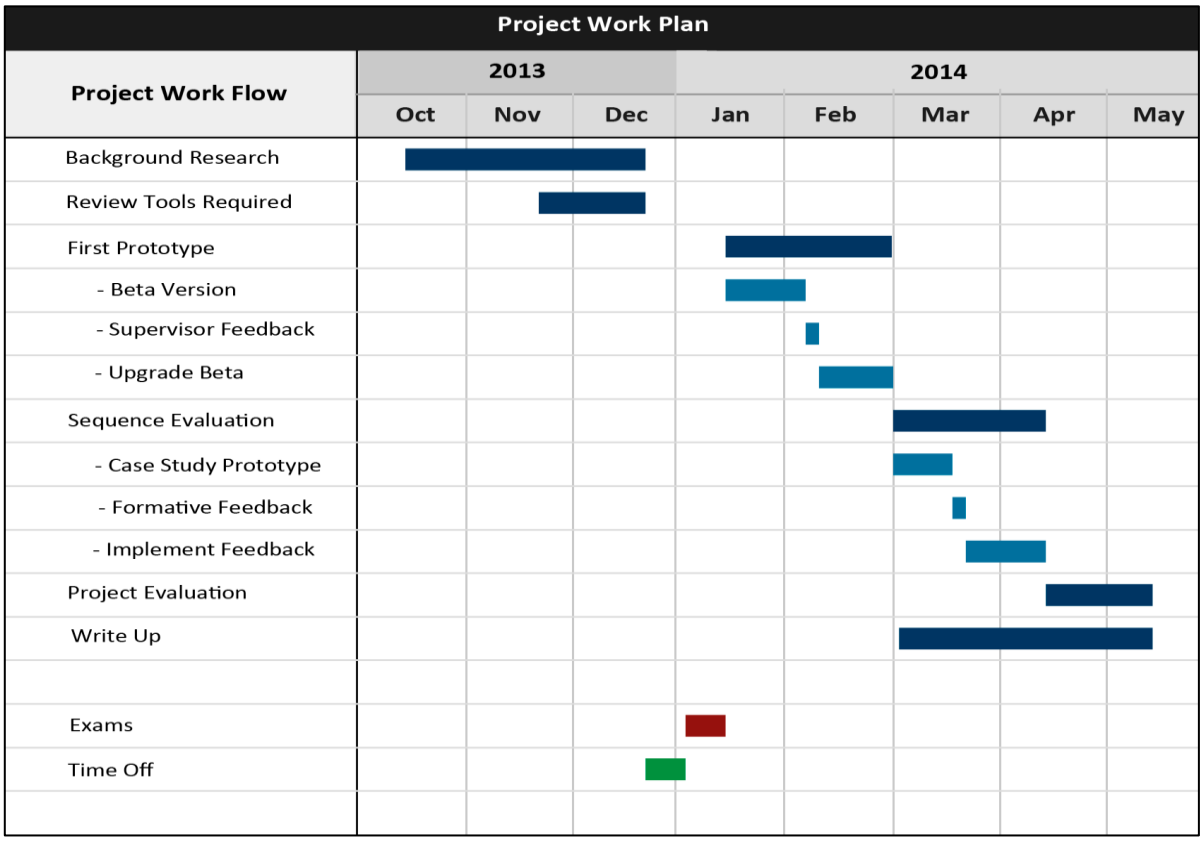


Figure 2.4.1 – The project schedule

3. BACKGROUND RESEARCH

This project required a lot of background research, particularly in the HTML5 framework. This was majorly due to the fact that the framework is comparatively new and in some cases has limited functionality.

3.1. SENSOR DATA

The sensor data had to be researched quite a bit as it required a detailed understanding of the various sensors and their data return types. After having done that, it required further research on how to use them and research including their precision and its effects: both positive and negative.

3.1.1. ACCELEROMETER SENSOR

This is the most widely used sensor amongst all sensors in a smartphone. This is because most games and apps that are dependent on motion detection make use of this sensor's data for their physics and function calls accordingly.

```
var ax = acceleration.x;
var ay = acceleration.y;
var az = acceleration.z;
```

The accelerometer sensor returns data on three axes: the x-axis, the y-axis and the z-axis. The data on these axes is usually very small and can return a positive or negative value, depending on which direction the device is being moved.

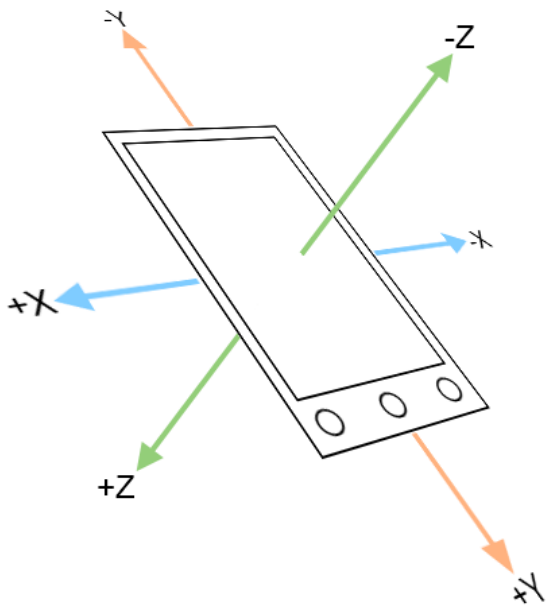


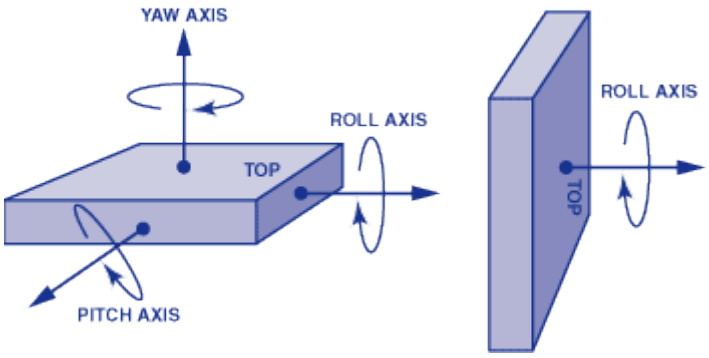
Figure 3.1.1.1 – Three axes of the accelerometer

The sensor data returned has the SI unit of **m/s²** across all axes. This data represents the speed with which the smartphone is accelerated in a particular axis. The data precision was of

importance for research purposes but had to be rounded up to be displayed to the novice user in an understandable manner.

3.1.2. ROTATION/GYROSCOPE SENSOR

The gyroscope is a device which rotates around three axes: the x, y and z axis. The rotation data for each of the three axes is returned as a value with the SI unit m/s2 and had to be converted to degrees for the understanding of the user.



3.1.2.1 – Gyroscope rotation around the x, y and z axis

In the modern smartphone, there is no specific gyroscope sensor. The rotation is instead calculated directly by converting the values of the accelerometer along the three axes into a degree value.

```
var ax = acceleration.x;
var ay = acceleration.y;
var az = acceleration.z;

var xAngle = Math.atan( ax / (Math.sqrt(ay^2 + az)));
var yAngle = Math.atan( ay / (Math.sqrt(ax^2 + az^2)));
var zAngle = Math.atan( Math.sqrt(ax^2 + ay^2) / az);

xAngle *= 180.00; yAngle *= 180.00; zAngle *= 180.00;
xAngle /= 3.141592; yAngle /= 3.141592; zAngle /= 3.141592;
```

This piece of code converts the accelerometer values into radians first, and the converts it into degrees. This value is then used in the visual representation and the precise value is printed out for the user to understand fluctuation. The value can be both positive and negative as rotation along the negative axes produces negative ax, ay and az values. The rotation has a max cap of 359^{o} because when the value is equal to 360^{o} the value has to be converted back to 0^{o} as both are the same point in space, with respect to the smartphone. This has to be particularly taken care of as otherwise; the gyroscope throws an error if this is not handled properly.

3.1.3. GEOLOCATION SENSOR

The geolocation sensor is the second most used sensor in a phone. But according to various discussions online, it has been widely considered as the most used sensor of all the sensors present in a smartphone.

The geolocation sensor returns a number of values. But these values are not in axes. Though in some cases, the longitude and latitude returns are considered as the x and y axis of the sensor and the altitude is considered as the z axis. However when returning, we need to explicitly call the value we are interested in and cannot just call the x, y or z value as in the case of the accelerometer sensor.

`position.coords.latitude`	–	Returns the current latitude
`position.coords.longitude`	–	Returns the current longitude
`position.coords.altitude`	–	Returns the current altitude
`position.coords.heading`	–	Returns the direction headed

The above lines of code show how the calling of values from the geolocation sensor is different from those of the accelerometer sensor.

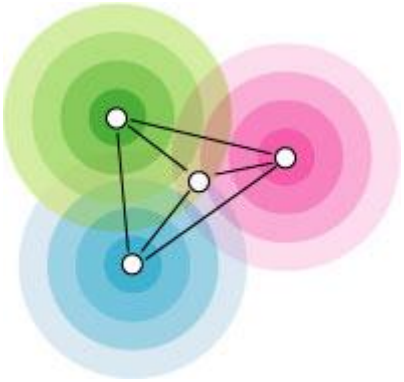


Figure 3.1.3.1 – Triangular position detection through grids scanning

The working of the geolocation sensor has been of particular interest. The sensor scans your position based on grids over the surface of the earth. It scans your position, and records it. Then, in order to be accurate, it scans the three nearest grids and verifies their neighbours. If all three grids have your position as their neighbour, then this position is returned, along with the longitude and latitude values. The most interesting aspect of the geolocation sensor is its ability to detect its error and warn the user of the accuracy of returned values. Despite sensors being very advanced, no smartphone or industrial geolocation sensor is void of errors. The margin of error is very low and hence, the accuracy is high.

3.1.4. LIGHT AND TEMPERATURE SENSORS

The light sensor is used a lot more in smartphones recently than the temperature sensor. The light sensor is also what is called the proximity sensor. It is majorly used to turn the screen off during phone calls to prevent heating, and to adjust the brightness of the screen in accordance with the level of light outside.

Light sensor works on three major scales and the SI unit is **lux**. The first scale is if: value < 50 lux. Then the result is dim and the screen's brightness is reduced. But, if: value == 0, then the screen is completely off. For if: value>50 lux && value<10000 lux, the light is deemed normal and the screen brightness is adjusted accordingly. For if: value>10000 lux, the light is deemed bright and the screen brightness is lowered considerably or raised higher, depending on the manufacturer's discretion.

The temperature sensor is rarely used as most weather apps depend on online weather providers and use API function calls to display weather data on the phone. The few cases where

temperature sensors are used are when an app specifically calls for it, or to check for overheating of the device.

The temperature sensor has been deemed obsolete by many app developers across many platforms. Though the support for the temperature sensors will be provided, it will no longer be updated on some platforms unlike the case with other sensors. The API provides the user across platform-specific frameworks, provide the user with function calls to access this sensor but it is unlikely to be implemented in cross-platform frameworks due to its limited use.

The light sensor and temperature sensor are expected to be combined into one API function call, but most developers are against this as the light sensor is very essential in most smartphone development in the modern smartphone era.

3.1.5. USER PRIVACY

While dealing with data as sensitive as a smartphone's sensors, the user privacy policy is regarded as one of the most important aspect of building an app that even remotely access, displays, stores or controls the sensor data.

The user privacy policy is very important as people hold their privacy in high regard and do not like to be ill informed when it comes to collecting their sensitive data. Though this data may not be as sensitive as a user's credit card, bank or social security details, the sensor data are equally sensitive as they can pinpoint the exact location of the user and if in the wrong hands, can be used to follow the user's actions discreetly. This is generally regarded as very precious data to any person having a deep knowledge of sensor data and knows how to decipher it to produce visual results.

Creating an app based on sensor data is very tricky as we have to inform the user of the sensitivity of the app accessing the user's smartphone sensors. Thus, this project ensures that none of the sensor data are publicly exposed and that the user is warned about the app being able to access the device's sensors and also explicitly states to the user that none of the data is being stored for display in a foolish manner. All variables used in the app have a local scope and cannot be accessed outside the particular functions. The functions themselves make use of the data just for representation and then discard them, when their call goes out of scope.

3.2. HTML5 DEVELOPMENT FRAMEWORK

The HTML5 application development framework is comparatively new in the market and is being sought after by many developers. This is largely due to the simplicity of its code and the new JavaScript functions added to it which makes accessing sensors and using sensor data a lot easier for the developer. Furthermore, it is a cross-platform framework, which lessens the hassle of creating multiple instances of the same app for each platform individually.

In Table 3.4.1.1, there is an in-depth comparison of the HTML5 mobile app development framework and some of the native mobile app development frameworks like the iOS development framework, Android SDK, Windows SDK and the BlackBerry Framework. The table further highlights in a summary, the pros and cons of the HTML5 development framework.

The image below gives a brief overlook of the pros and cons of the HTML5 framework over the native development framework:

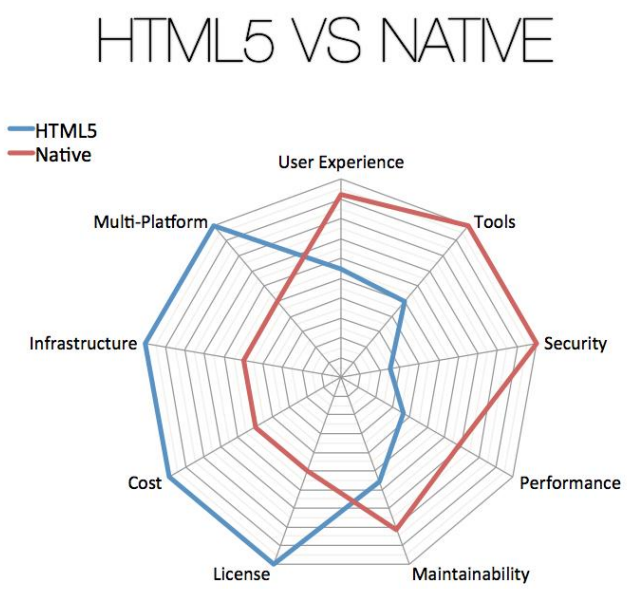


Figure 3.2.1 – HTML5 vs Native app development frameworks

Unlike its predecessor, HTML5 does not only contain the basic HTML attributes and tags. HTML5 is an entity that includes HTML5, CSS3 and JavaScript3.x, which makes it a much more powerful and versatile mark-up/scripting language to write complex applications in a simple manner.

3.2.1. PROS OF HTML5

- **HTML5 is cross-platform.** It involves a hassle free, one instance production for multiple mobile operating systems.

- **HTML5 involves simple tags.** All the tags used in HTML5 are basic HTML4 tags and some new added tags which have been explicitly explained in the documentation. Their usage is fairly simple and reduces a lot of scripting involved.

- **It has a lot simpler JavaScript functions calls added.** Apart from being very simple as a language, HTML5 has made additions to the extensive JavaScript library, by adding quite a few new functions calls. The majority of them are concerned with sensors and their accessibility, one of the main reasons this project was based on HTML5.

- **It is very cost effective.** When cost is mentioned in this scope, it doesn't only include finance, but also in terms of labour, hours required and lines of code. HTML5 overall, is very cost effective. Financially, it practically costs nothing to build. Resourcefully, it requires far less labour and working hours to build an app and the lines of code are considerably less.

- **Licensing is not a problem.** Considering that this project used sensor data, researching online threw out astronomical prices to access and use sensor data in terms of certificates and user privacy policies. HTML5, on the other hand, allows the developer to use sensor data whichever way he/she pleases, provided the developer explicitly

specifies what the intended use of the data is and warn the user of exposing sensor data to the app.

3.2.2. CONS OF HTML5

- **Limited security.** HTML5, though very versatile, lacks the security measures that the native application development framework provides. Since it is a combination of mark-up and scripting language and JavaScript in itself has very limited security measures available, HTML5 is usually unsuitable for applications dealing with overly sensitive data.

- **Limited maintainability.** Native application frameworks win over HTML5 simply in terms of the ease with which you can maintain them and push out content and maintain and update existing content. HTML5 is very limited in this aspect as unless you use local storage, which doesn't last for an indefinite amount of time, you cannot maintain, record and update data without releasing multiple versions of the app as updates for every small update and bug fix.

- **User interaction is not quite smooth.** App users love a neat and smooth user interface. Though HTML5 and JavaScript do provide a certain level of smoothness to the user interaction, it still doesn't compare to the extensive options and customizations available in the native app development framework.

- **Availability of tools is limited.** HTML5 does not have the massive load of tools and plugins except the additions JavaScript libraries and modules. Though these modules are very interactive and perform pretty decently, they do not match up to the countless tools and plugins available in the native app development library.

- **Limited existence of APIs.** The main problem of HTML5 and JavaScript is that due to its lack of security, no modern browser or mobile app development framework supports file writing, reading and storage. The new File System API of HTML5/JavaScript is very limited and is not nearly as efficient as the File System APIs of the native app development library.

- **Limited sensor accessibility.** Being new to the development scene, the HTML5 API functions do not provide access to the innumerable sensors present in the modern smartphone.

3.2.3. HTML5 SENSOR DATA FRAMEWORK

The HTML5 sensor data framework is very limited as the language is still under development and is not as comprehensive as the native app development frameworks. Currently, the HTML5 development framework only supports access to the accelerometer and geolocation sensors. This is a massive drawback as application developers working with sensors usually require access to almost all the sensors present in the modern smartphone to be able to make their app effective and evaluative against apps built on the native framework.

The limited sensor accessibility has been cited as the main problem for developers choosing native app development frameworks over HTML5. The only HTML5 compliant web app development framework which supports access to multiple sensors is the Mozilla HTML5 Dev Framework (M5DF). But the function calls in the sensor APIs are currently not supported by

most major browsers except Mozilla Firefox, Mozilla Firefox Geko and Mozilla OS. Though these APIs are under consideration with the World Wide Web Consortium (W3C), it is expected that they will be soon available to developers within the next 2 to 3 years.

3.3. IMPLEMENTATION TECHNOLOGY

The implementation technologies chosen for this project have been duly considered and compared with one another before being confirmed as the basis of the development of this app. There were quite a few options but the option most relevant with the development of an app that used and displayed sensor data was chosen. Some of the most popular choices are compared here, with each of them being explained and finally the explanation on the language chosen is mentioned. The criteria were narrowed down to two main scripting languages: JavaScript and PHP script.

3.3.1. JAVASCRIPT

JavaScript is the primary scripting language of HTML5. The latest version of the script, JavaScript3, was released along with HTML5 and boasts of a whole lot of function calls that allow developing an application using the framework a whole lot easier and provides the much needed extended versatility to the already versatile mark-up language.

The new JavaScript3 (hereafter: JavaScript), comes with HTML5 app development specific functions which make the tasks of establishing sensor connections, closing sensor connections and using sensor data very much simpler. For example:

```
function contactAcc (acceleration) {
        var ax = acceleration.x;
        var ay = acceleration.y;
        var az = acceleration.z;
        var element = document.getElementById('acc');
        element.innerHTML = 'Acceleration X = ' + acceleration.x + '<br />'+
'Acceleration Y = ' + acceleration.y + '<br />' + 'Acceleration Z = ' +
acceleration.z;
}
```

This particular piece of code is a simple function using the JavaScript, which takes acceleration as a function parameter. Upon being called the function prints the x, y and z-axis values from the accelerometer to a division `<div id='acc'>` in the body of the HTML5 page.

JavaScript makes connecting to the sensors in the smartphone easier through pre-defined variables like `acceleration` (for the accelerometer), `position` (for geolocation) and `compass` (for direction).

Though the accessibility is limited, the implementation is quite straightforward and is explicitly declared in the documentation.

3.3.2. PHP SCRIPT

PHP script is one of the most major scripting languages used across most modern websites. This is due to its flexibility and its ability to combine different scripting languages within itself and make use of their functions and tools as its own.

Apart from being largely versatile, PHP has a lot of security measures for uploading, downloading and transmitting sensitive data such as the data from smartphone sensors.

But the cons of PHP over JavaScript is, it does not have any API function calls to explicitly establish a connection to the sensors and relies on JavaScript interweaving to connect to the sensors. This means even if you use PHP, the basic connection to the sensors and the data handling, is done by JavaScript as an interwoven mesh inside the PHP script. This makes little sense as it does not increase data security but only complicates the code. Apart from this problem, PHP is a server-side scripting language which in this case, considerably slows down the application speed overall.

3.3.3. HTML5/CSS3

HTML5/CSS3 is the primary mark-up and styling language combination for HTML5 mobile application development. This is due to the simplicity of the HTML5 mark-up language and CSS3 as a very versatile styling language with its new additions to work wonders without creating an image for every little effect needed. This is particularly useful in the displaying of sensor data as visualizing sensor data requires a lot of versatility in styling because the explicit styling is used to differentiate between two sensors and between the different data of a particular sensor.

3.3.4. CONCLUSION

After a considerable comparison between JavaScript and PHP script, it was concluded that JavaScript suited the needs of this project aptly enough for PHP to be disregarded. Even though PHP script was a very viable choice, the fact that it had no direct API function calls to access sensor data and relied on interwoven JavaScript to carry out this task makes obsolete in terms of practicality. It will just increase the lines of code needed to create the exact same output but with more complicated methods and function calls.

3.4. EXISTING APPLICATIONS

Applications using sensor data aren't anything new in the app stores across various mobile devices. There have been a lot of applications showcasing what sensors can do and even displaying sensor data. Some of the popular ones are ***AndroSensor*** for Android and ***Sensor Data*** for iOS.

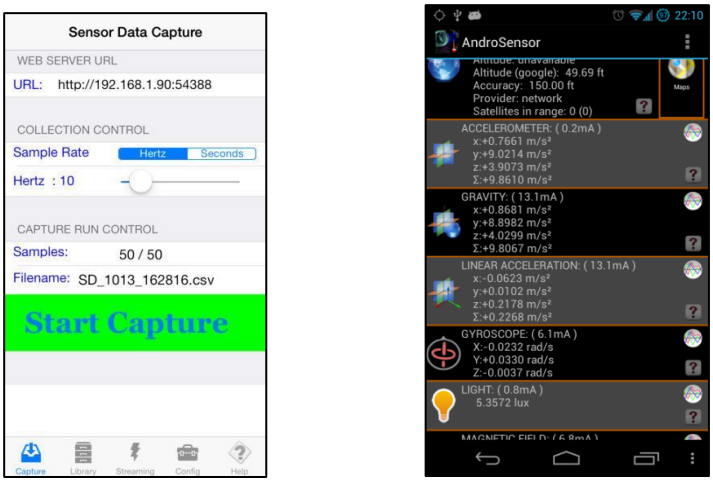


Figure 3.4.1.1 and Figure 3.4.1.2 – Sensor Data for iOS (l) & AndroSensor for Android (r)

3.4.1. EVALUATION OF EXISTING APPLICATIONS

With the growth in the number of apps using sensors, there has also been a rise in the number of apps that display raw sensor data. Some of the most popular ones are the Sensor Data for iOS and the AndroSensor for Android. A screenshot of both these apps are show below:

Most applications that display sensor data, like the ones above, simply display the raw data and have no means to visualize it. But what they do have is the ability to record data. This is one aspect that sensor applications built in HTML5 lack, as the File System APIs in HTML5 are not allowed writing privileges due to lack of ample security measures. Though JavaScript provides an API to write files, it is blocked by most modern browsers and mobile devices, again, due to security issues. Below is a table that highlights some key aspects of the native apps with comparison to this project.

Features	HTML5	iOS	Android	Windows	BlackBerry
Accelerometer	✓	✓	✓	✓	✓
Gyroscope	✓	✓	✓	✓	✓
Geolocation	✓	✓	✓	✓	✓
Light	×	✓	✓	✓	✓
Temperature	×	✓	✓	✓	✓
File Storage	×	✓	✓	✓	✓
Visualization	✓	×	×	×	×
Security	×	✓	✓	✓	✓
Smooth UI	✓	✓	✓	✓	✓
Tools	×	✓	✓	✓	✓
Cost Effective	✓	×	×	×	×

Table 3.4.1.1 – Comparison between HTML5 and native development frameworks

3.4.2. CONCLUSION

A thorough analysis of the existing native app development framework applications and HTML5 app development framework applications reaches one conclusion. An application built on a combination of HTML5 and native framework is the best solution to building a sensor app that has all the versatility and visualization power of HTML5, but at the same time has ample security along with recording capabilities of the native framework.

3.5 HUMAN INTERACTION

The simplicity in the human interaction aspect of a sensor data app built using HTML5 is the primary source of the preference of the new app development framework over the native app

development framework. The interface for this project is simple and concise. It is meant to be appealing to the user and help the user to easily navigate their way around the application.

3.5.1. USER INTERFACE

A lot of research was conducted into the project's user interface as users disliked complicated layouts and too much navigating at the cost of being fancy. The main aim of the design aspect of this project is to keep the design minimal and yet bold, using big icons and images, along with a dark on light combination. Another main reason for using a dark background is because a dark colour is proven to reduce battery consumption drastically, over an app with a light background. This is particularly effective and helpful whilst trying to make an app that displays sensor data and uses minimal power and optimises on speed, performance and power consumption.

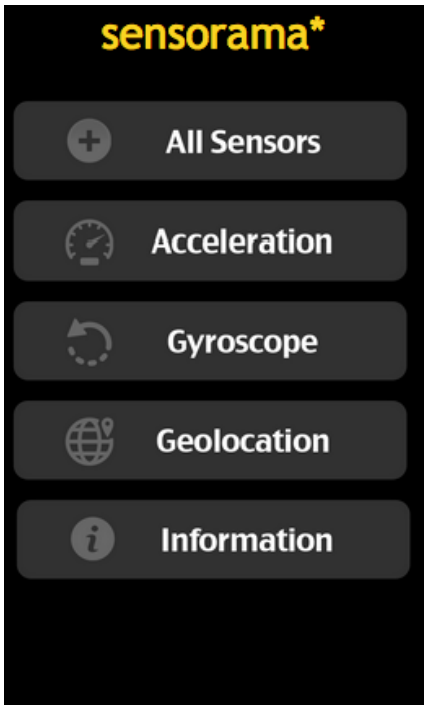


Figure 3.5.1.1 – A screenshot of the main menu of this project

3.5.2. DESIGN PRINCIPLES

The main design principles followed for this project are: simplicity, maximum visibility, smart use of available space to display all the available information in a concise manner and a colourful outlook to sensor data.

The user interface has been carefully designed from scratch. Considering the fact that minimal is good. Another important aspect of the design is the smart usage of available space. This project has taken into account that mobile devices do not have infinite space and hence the project has been broken up into separate sections and also a combination of all the sections, giving the user a wide range of options to view data.

The information displayed is very concise and also highly precise. No compromise has been made on the precision of the data as research has indicated that sensor data visualization apps need to focus on precision over flashiness and overly exuberant images and formats.

3.6. EVALUATION TECHNIQUES

A number of evaluation techniques were used to ensure that this project was able to achieve the maximum number of the aforementioned objectives. These include:

3.6.1. FOCUS GROUP INTERACTIONS

A number of interactions were conducted with the focus group of this app, which mainly included people working on sensor data and those interested in starting a work related to this field. Going online, I found a private Google Group "Sensors Unlimited", which has over a 100 members. Upon uploading a rough draft of this app, I asked anyone willing to test this app on their respective phones. After quite a few interactions and user feedback, this application was tweaked and remade to suit the needs of users who have already previously worked with other sensor data visualization apps.

3.6.2. FACE TO FACE DISCUSSIONS

A number of face-to-face discussions were held with people within my social group, who had no knowledge of sensor data or some who knew about sensor data but didn't know what it symbolised. These social groups were offered a try at the app and their feedback regarding the beta version of this app was collected and the same social group was shown the finished product, which was duly updated with the initial feedback provided. There were a total of 6 people in the social group who tried the app, and 5 of them gave feedbacks.

4. SYSTEM DESIGN

The analysis of the various requirements of the system in order to get the resultant design is described in this section. The solutions discussed within are taken into consideration in the implementation of the application, which is discussed in the next section.

4.1. REQUIREMENTS

The identified requirements of the project are listed below:

1. The project should provide a visual representation of sensor data.
2. The app should specifically highlight the different values output by the sensor in the graphical view.
3. It should update this data and visualization in real time.
4. The app should take care of the precision of the data.
5. The app should take display the visual and graphical aspect of the data.
6. The app should have a neat section for each sensor type that is accessible apart from a combined sensor data display section.
7. The app should explicitly update the maps only when the user's geolocation data has undergone changes and not otherwise.

4.2. HIGH LEVEL DESIGN

To start designing a system, its major components must be identified along with the component's connectivity, to produce a high-level design. Providing real time graphical update of the data is the key aspect of this project. Therefore the application has to tie in the data received with the graphical rendering of the data, so that the visualization and the data returned are both in real-time and are always in sync. The first function is the **`success<sensor-name>`** function. This function is called upon by the application in the **`window.onLoad`** process or simply put, as soon as the app loads. The first call to this function sees the real-time values for the respective sensor stored in local variables. These values are the passed on to their respective **`<span>`** ids. Before the function terminates, there is a call to the chart renderer function. This function takes in the same parameter as the success function. All the sensor data values are passed to the chart renderer, which renders the chart in its respective **`chart_div`**. When the function finally ends, the real-time values and the real-time graph are all rendered in their respective divisions on the page of the app.

The below diagram gives a simple step-by-step view of the system design and the working of the real-time data updating and chart rendering done by the application, the two major contents of this project.

The image also implicitly states that sensor data precision is not altered at any point of time. Whatever data the sensor returns, is directly output to the application divisions.

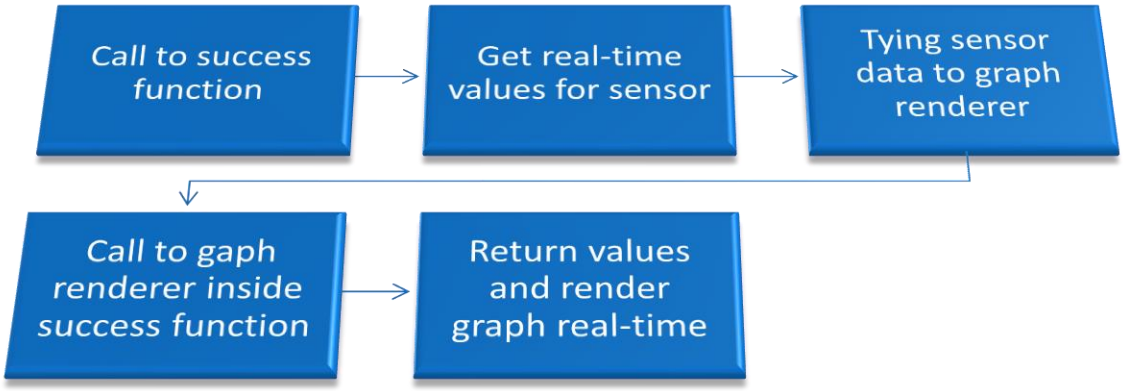


Figure 4.2.1 – Hierarchy in system design

4.3. AUTOMATIC DATA UPDATE

The basis of this application is to provide real-time sensor data updates. This is done through the success function. There are explicitly designed success functions for each sensor in each individual section of the program. But, in the "All Sensors" page which contains all the sensor data collectively, there are just two functions declared as the project required multiple sensor data values to be updates simultaneously.

4.4. DATA PRECISION

Data precision is very vital for an app displaying data from mobile sensors. Hence, every effort has made in this program to display the raw values returned by the sensors and use the same raw values to render the visuals, rather than alter them in any way to suit the needs of this application. The data is not rounded and the visuals are made such that if the value returned is negative, they are not converted to a positive value but instead the graph renders in the negative vertical half (bottom half) of the graph without altering its central position on the graph.

4.5. GRAPHS GENERATOR

As with the case of the success functions for each individual sensors, in each section dedicated to a particular sensor, there is also a dedicated graph renderer function explicitly specified for that sensor. This function takes in the real-time sensor data called in the success function as its input and renders the chart in its respective division in the sensor page. This function is called inside the success function and hence updates real-time as the success function updates, every time there is a chance in the sensor data value.

4.6. MAPS GENERATOR

The maps generator is a function that acts as the chart render for the geolocation sensor. Since there is no chart to be rendered for the geolocation sensor, the application renders a map, pin-pointing the exact location of the user instead. The map generator has a function that sets the street view image of the user's current location as the app background. In the geolocation

sensor page, this function is called inside the success function. The success function takes in the geolocation data of the user from the geolocation sensor and draws a map in the respectively specified canvas. In other sensor pages, the function to set the street view is called instead, as a geolocation success function in the **`window.onLoad`** property of the page.

5. IMPLEMENTATION

The implementation process is the process of software engineering after the designing process, wherein the designs analysed are implemented into the actual project.

The implementation was carried out in various phases. Each phase of the implementation is discussed below in detail along with the code designed to render the implementation.

5.1. IMPLEMENTATION OF THE ACCELEROMETER DATA READING

The accelerometer data can be accessed in the HTML5 app development using the `acceleration` keyword. The accelerometer returns data values across 3 axes, namely the x-axis, the y-axis and the z-axis. Each of these values can be individually accessed by appending the respective letter of the axis after the `acceleration` keyword.

```
        acceleration.x          // x-axis value of the accelerometer
        acceleration.y          // y-axis value of the accelerometer
        acceleration.z          // z-axis value of the accelerometer
```

The implementation is quite straightforward. The accelerometer page has a success function that is called every time the value of the accelerometer value changes.

```
function successAcc(acceleration)
{
        var acc;

        acc = document.getElementById("accx");
        acc.innerHTML = acceleration.x;

        acc = document.getElementById("accy");
        acc.innerHTML = acceleration.y;

        acc = document.getElementById("accz");
        acc.innerHTML = acceleration.z;
```

The above code snippet is the implementation of the accelerometer success function. This function updates the values for the divisions **accx**, **accy** and **accz**, according to the x, y and z axis values returned by the accelerometer, every time the value for the accelerometer changes.

This is handled by the **watchAcceleration** function provided by the HTML5 mobile application development framework and works as follows:

```
try {
        intel.xdk.accelerometer.watchAcceleration(successAcc, options);
}
catch (error) {
        alert("Accelerometer error: \n" + error.name + ": " +
error.message);
}
```

This is defined inside the **`onDeviceReady()`** function, which is called when the device is ready.

This function collects the data from each axis of the accelerometer, as shown in Figure 3.1.1.1, and stores them in a variable. These variables are then printed to the respective divisions stated above. This happens for every change in the accelerometer readings across all three axes as we are watching the acceleration for change, every millisecond.

5.2. IMPLEMENTATION OF THE GYROSCOPE DATA READING

Modern smartphones do not have an individual gyroscope sensor. Instead they use the accelerometer sensor reading and convert the readings into degrees for getting the gyroscopic values in degrees.

The math for this particular application is pretty straightforward and simple:

```
        var ax = acceleration.x;
        var ay = acceleration.y;
        var az = acceleration.z;

        var xAngle = Math.atan( ax / (Math.sqrt(ay^2 + az)));
        var yAngle = Math.atan( ay / (Math.sqrt(ax^2 + az^2)));
        var zAngle = Math.atan( Math.sqrt(ax^2 + ay^2) / az);

        xAngle *= 180.00; yAngle *= 180.00; zAngle *= 180.00;
        xAngle /= 3.141592; yAngle /= 3.141592; zAngle /= 3.141592;
```

The <axis>Angle variable, takes the raw accelerometer data and converts it into the respective degree value through a series of simple calculations. Once these calculations are done, the resultant values are passed as the gyroscope values. Below is an implementation of the logic:

```
function successAcc(acceleration)
{
        var element;
        var ax = acceleration.x;
        var ay = acceleration.y;
        var az = acceleration.z;

        var xAngle = Math.atan( ax / (Math.sqrt(ay^2 + az)));
        var yAngle = Math.atan( ay / (Math.sqrt(ax^2 + az^2)));
        var zAngle = Math.atan( Math.sqrt(ax^2 + ay^2) / az);

        xAngle *= 180.00; yAngle *= 180.00; zAngle *= 180.00;
        xAngle /= 3.141592; yAngle /= 3.141592; zAngle /= 3.141592;

        element = document.getElementById('gyrox');
        element.innerHTML = xAngle;
        element = document.getElementById('gyroy');
        element.innerHTML = yAngle;
        element = document.getElementById('gyroz');
        element.innerHTML = zAngle;
        drawChart(acceleration); }
```

In the code snippet, the accelerometer data are converted as mentioned above and the converted data is passed to their respective divisions. This success is called in the gyroscope section every time there is a change in the accelerometer reading. Exactly similar to the way the accelerometer success function was called in section 5.1.

5.3. IMPLEMENTATION OF THE GPS DATA READING

The geolocation data in the modern smartphones are accessed through the keyword **position**. Using this keyword, we can get individual values that the sensor returns. For extraction of these values, we simply append **.coords.<longitude / latitude / altitude / heading>** to **position**.

The following code snippet is the implementation of the geolocation data reading from the sensor:

```
//Part 1
function successGeo(position)
{
        var element;
        var direction;

        var compass = document.getElementById('compass');

        if(position.coords.heading===0 || position.coords.heading === 360)
                direction = "North";
        if(position.coords.heading> 0 && position.coords.heading < 90)
                direction = "North East";
        if(position.coords.heading === 90)
                direction = "East";
        if(position.coords.heading>90 && position.coords.heading < 180)
                direction = "South East";
        if(position.coords.heading === 180)
                direction = "South";
        if(position.coords.heading> 180 && position.coords.heading < 270)
                direction = "South West";
        if(position.coords.heading === 270)
                direction = "West";
        if(position.coords.heading>270 && position.coords.heading<360)
                direction = "North West";

        element = document.getElementById('geo1');
        element.innerHTML = position.coords.latitude;

        element = document.getElementById('geo2');
        element.innerHTML = position.coords.longitude;

        element = document.getElementById('geo3');
        element.innerHTML = position.coords.altitude;

        compass.innerHTML = direction;
        setFrame(position);
```

There are a couple of parts to success function of the geolocation sensor, which shows how the geolocation data is read from the geolocation sensor and how the map is draw in the division

using Google Maps API. For the reading of sensor data, this section focuses on Part 1 of the code. Part 2 will be discussed in section 5.5.

The above snippet is Part 1 of the two-part code. This part handles the reading of the sensor data from the sensor. This project uses only the **`latitude`**, **`longitude`**, **`altitude`** and the direction (**`heading`**) of the geolocation sensor. These are the main returned data values of the geolocation sensor and a sensor data visualization app is interested majorly in them. These data values let the developer do a variety of implementations.

The values here are not stored in any local variable, as we want these values to be as secure as possible, solely using JavaScript. Hence, we pass the position as a parameter to the success function of the geolocation sensor and use the API object calls to get the values and pass them directly to the respective divisions in the geolocation sensor page.

The downside to this is that the geolocation sensor in the mobile devices is relatively slower than the accelerometer sensors and hence takes a little while to load the data values and update the map. Even though the updates are instantaneous and take place as soon as there is a change in the geolocation values, it may seem that the update is delayed and that the values are not real-time.

With respect to using the direction (**`heading`**) of the geolocation sensor, the number should not be displayed to the user as these numbers are not relevant and a sensor app visualizer is more bothered about the 'true' direction which is more appreciated as a string rather than a number. For this case, we perform the if-condition with the heading and set directions for every value. The values are particularly checked at 0 and 360 as at these values, if there are no checks performed, the geolocation sensor has been known to throw unexpected NULL errors.

5.4. IMPLEMENTATION OF REAL TIME MAP GRAPHS

The real time graphs are the most important part of this project apart from the precise manner in which the raw values are displayed. This project combines the both to give a very powerful visualizer which not only gives a graphical view of the sensor data for easier comparison, but also prints out the raw value as a supplement to the graph.

The real time graphs are rendered using the Google Visualization API. For the visualization API, the project uses the below mentioned line of script included in the head of the pages where it is used:

```
<script type="text/javascript" src="https://www.google.com/jsapi"></script>
```

Once this has been included, the project contains dedicated **`drawChart`** functions for each individual sensor, but a combined **`drawChart`** function for the "All Sensors" page. Below are the dedicated functions for every sensor and a brief explanation about them.

ACCELEROMETER GRAPH RENDER FUNCTION

The code snippet that follows takes in the **`acceleration`** keyword as a parameter, and stores the accelerometer data values for all 3 axes as local variables, which are the passed to the Google Visualization API as array objects.

The code snippet:

```
google.load("visualization", "1", {packages:["corechart"]});
google.setOnLoadCallback(drawChart);

function drawChart(acceleration) {

 var ax = acceleration.x;
 var ay = acceleration.y;
 var az = acceleration.z;

        var data = google.visualization.arrayToDataTable([['T', 'X', 'Y',
'Z'], [acceleration.getTime, ax, ay, az]]);

        var options = {
        backgroundColor: 'transparent',
        legend: {postion: 'top',
        textStyle: { color: '#fff' }
        },
        hAxis: { textStyle: { color: '#fff' } },
        vAxis: { textStyle: { color: '#fff' } }
        }

        var chart = new
google.visualization.ColumnChart(document.getElementById('chart_div'));
        chart.draw(data, options);
}
```

The two lines above the function initialize the Google Visualization Library and set the call back function when the window loads. The **drawChart(acceleration)** function is the main chart rendering function for the accelerometer sensor data. The variable **data** accepts the x, y and z-axis values of the accelerometer sensor data as array objects and plots stores them in a tabular form. The variable **options** sets the miscellaneous options for the look and feel of the graph. This function is called inside the success function of the accelerometer, so that every time the accelerometer value changes, the chart is redrawn in **chart_div** division.

GYROSCOPE GRAPH RENDER FUNCTION

The code snippet that follows, renders the graph in the gyroscope page whenever the value of the gyroscope changes. The working of this function is exactly similar to that of the previous function as both take in **acceleration** as parameters and return acceleration x, y and z values. In this case, we change the first three lines of the **drawChart** function to this:

```
        var ax = acceleration.x;
        var ay = acceleration.y;
        var az = acceleration.z;

        var xAngle = Math.atan( ax / (Math.sqrt(ay^2 + az)));
        var yAngle = Math.atan( ay / (Math.sqrt(ax^2 + az^2)));
        var zAngle = Math.atan( Math.sqrt(ax^2 + ay^2) / az);

        xAngle *= 180.00; yAngle *= 180.00; zAngle *= 180.00;
        xAngle /= 3.141592; yAngle /= 3.141592; zAngle /= 3.141592;
```

For the variables to be passed, we pass **xAngle**, **yAngle** and **zAngle** to **data**.

5.5. IMPLEMENTATION OF REAL TIME MAP UPDATE

This section deals with the implementation of drawing the map in the geolocation page in real-time. As discussed earlier, the success function of the geolocation sensor is made up of two parts. The first part, discussed in section 5.4, handles the reading of geolocation data in real time. This section deals with Part 2 of the code, which handles the drawing of the map in real time.

To draw the map, the project uses the Google Maps API. This line of script is included in the head section of the geolocation page:

```
<script src="//maps.googleapis.com/maps/api/js?v=3.exp&sensor=true">
</script>
```

The function code (Part 2 of success function of geolocation) is as follows:

```
//Part 2
(function() {

if(!!navigator.geolocation) {

        var map;

        var mapOptions = {
        zoom: 15,
        mapTypeId: google.maps.MapTypeId.ROADMAP,
        draggable: false,
        zoomControl: false,
        scrollWheel: false,
        disableDoubleClickZoom: true,
        disableDefaultUI: true
        };

        map = new google.maps.Map(document.getElementById('google_canvas'),
mapOptions);

        navigator.geolocation.getCurrentPosition(function(position) {

        var geolocate = new google.maps.LatLng(position.coords.latitude,
position.coords.longitude);

        var marker = new google.maps.Marker({
        map: map,
        position: geolocate,
        });
        map.setCenter(geolocate);
        });

        } else {
        document.getElementById('google_canvas').innerHTML = 'No
Geolocation Support.';
}

})(); }
```

This is a reference to the Google Map API. This function is called inside the success function and creates a map on the geolocation acquired in the geolocation success function. The values acquired are passed on to the Google Map API function call, which the draws the map in the **google_canvas** division, whenever there is a change in the user's geolocation.

5.6. IMPLEMENTATION OF REAL TIME STREET VIEW UPDATE

The Google Street View real time update uses the Google Street View API. The API is a simple link. Every sensor page contains an iFrame which covers the entire background and does not scroll.

```
<iframe src="load.html" id="mapframe">iFrame unsupported</iframe>
```

The above snippet creates an iFrame whose styling is controlled by CSS entities to the id **mapframe** in the stylesheet. The function **setFrame** takes in **position** as the parameter and sets the value of the iFrame to the current street's image, using the Google Street View API. The function **setFrame** is as follows:

```
function setFrame(position) {

 var element = document.getElementById('mapframe');
 element.src = "http://maps.googleapis.com/maps/api/streetview?size=" +
element.clientWidth + "x" + element.clientHeight + "&location=" +
position.coords.latitude + "," + position.coords.longitude + "&heading=" +
position.coords.heading + "&pitch=-0&sensor=true";

}
```

The **setFrame** function is called in every the accelerometer and gyroscope sensor pages instead of the geolocation success function, but in the geolocation and all sensors pages, it is called inside the geolocation success function.

5.7. IMPLEMENTATION OF THE GUI

The GUI is implemented using CSS3. The style.css in the header handles the customization of objects inside the body of the pages.

Below are the major snippets of the CSS code which handle particular objects within the body:

```
body {
 margin:0;
 padding:0;
 font-family: Calibri;
 background: #000;
 color: #FFFCDD;
 font-size:18px;
 width:100%; }
```

The above snippet handles the styling properties of the body tag. The **body** is the main holder of all the content within the page.

```
a:link, a:visited, a:hover, a:active { color: #fff; text-decoration: none;
}
```

The snippet above, handles the links within the page. It sets the colours and text decoration properties of links.

The following CSS lines handle the alignment of **content** held within the body. The **boundingbox** class managed the styling for the paragraph holding span divisions for the values of the sensor data.

```
content {
 width:inherit;
 position:absolute;
 top: 0;
 left: 0;
 z-index: 1;
}

h1 {
 border:0;
 background: #000;
 width: inherit;
 font-size: 18px;
 padding-left:5px;
 padding-top:5px;
 padding-bottom:5px;
 color: #FDEFD3;
 text-align:left;
 margin-top:7px;
 margin-bottom: 5px;
}

.boundbox {
 padding-left:15px;
 text-align:left;
 color: #fff;
}
```

Another important CSS styling is the **logo**, which handles the main logo at the top of every page and the **positioning of divisions for every sensor data which are explicitly specified and are unique**.

```
logo {
 font-family:Trebuchet MS;font-size:40px;
 width:inherit;
 padding:5px 5px 5px 5px;
 color: #F9D423;
 text-align: center; display: block; }
```

```
#accx, #accy, #accz,
#geo1, #geo2, #geo3, #compass,
#gyrox, #gyroy, #gyroz, #light
{ float:right; padding-right:15px; }
```

Finally there are just two more major CSS elements left, the **mmenu** class, which handles the main menu, and the **iframe** style modifier which directly modifies the styles for the iFrame tag as we have only one iFrame throughout the whole application.

```
.mmenu img {
 width: 95%;
 vertical-align:middle;
 padding: 6px;
}

.mmenu {
 font-size:25px;
 text-align:center;
 width: 100%;
}

iframe {
 width:100%;
 border:0;
 height:100%;
 position:fixed;
 z-index: -1;
 opacity: 0.5;
}
```

6. TESTING

Testing is a very important aspect of software engineering. Testing informs the developer of the bugs present in the application and gives him/her an opportunity to rectify the errors before pushing the application out to the public. Testing is usually a drawn out process and is done multiple times, for multiple cases. The main resolve of testing as known to computing scientists is: "Test your application till it breaks. If it does not break, you Sir, have done an excellent job!"

The testing for this application was carried out in various phases. The phases are discussed in this section.

6.1. EMULATOR TESTING

Since the availability of devices was scare, majority of the pre-beta testing was done on the Android, iOS and Windows Phone emulators. This included trying it on devices of various screen sizes and processing powers and trying to get the desired output.

Below is a table scoring the application performance out of a score of 10 on some popular devices. This table has been generated using scoring results from the emulator on the following traits.

Please note: There is a problem with HTML5 in iOS. Though auto-rotation of this application has been disabled, iOS automatically auto-rotates the app. This is not a glitch of the application as this is a well-known within the Intel XDK community and is being addressed currently.

Traits	Android	iOS	Windows	Others
Speed	8/10	9/10	7/10	6/10
Precision	10/10	10/10	10/10	8/10
Power	7/10	6/10	6/10	8/10
Usability	8/10	8/10	7/10	4/10
Interaction	8/10	9/10	6/10	6/10
Smoothness	9/10	8/10	7/10	4/10
Lightness	8/10	6/10	8/10	6/10
Crash Prone	8/10	8/10	7/10	3/10

Table 6.1.1 – Traits scored by the project on various emulators

6.2. REAL TIME TESTING

After extensive emulator testing the app was distributed to users as a Beta and verbal and casual feedback from the users were collected. Real time testing included posting the app on sensor data related groups and passing it around social groups.

6.3. DATA PRECISION TESTING

Since this application deals with sensor data, the precision of the data involved is of utmost importance. For this purpose the sensor data is carefully calculated and for the graph drawing as well as raw data display, the direct raw values from the sensors are used to avoid any data deficit in terms of precision. All the data is maintained until the maximum available decimal digit the sensor provides the data as.

6.4. USER INTERFACE TESTING

The user interface has been tested for bugs in terms of link breaks and image breaks. It has also been tested for size and whether it perfectly fits in any phone size. For this matter, all the sizes used in the user interface are in percentage so that it adjusts according to the screen size to avoid any “ugly UI” errors.

7. PROGRAM EVALUATION

After the Beta tests and user feedbacks gathered, the program was updated with the recommendations and released as an Alpha.

7.1. EVALUATE TEST RESULTS

The testing provided a lot of bugs in the Beta mode. Some of the major bugs were:

- The sensor pages didn't adjust according to the screen size but appeared as full-sized web pages.
- Every time the graph rendered, the application would shift focus to the graph, thus making viewing the data on the lower end of the page difficult.
- Street view would not update on some devices as they application was not being granted appropriate permissions while installing.

These errors were rectified in the following ways:

- Intel XDK development platform was not being initialised properly and the JavaScript Bridge kept collapsing due to which the page was rendered as a basic HTML page and not as an HTML5 application.
- The options value passed to the success functions rendered the same graphs repeatedly as they frequency was set. The frequency was set to 1 millisecond, but the graph frequency was set to detect change in sensor data and only then render.
- API key specified was withdrawn and data supplement via sensors was enabled, thus allowing the sensors to access the Google Street View API and thus adding it as permission for the app while installing.

7.2. USER FEEDBACK

As mentioned in section 3.6.3, six people volunteered to test this application after they were duly briefed of all the data this application could access. Out of the 6, 5 of them left a feedback on both, the Beta and the Alpha version.

Beta version feedback:

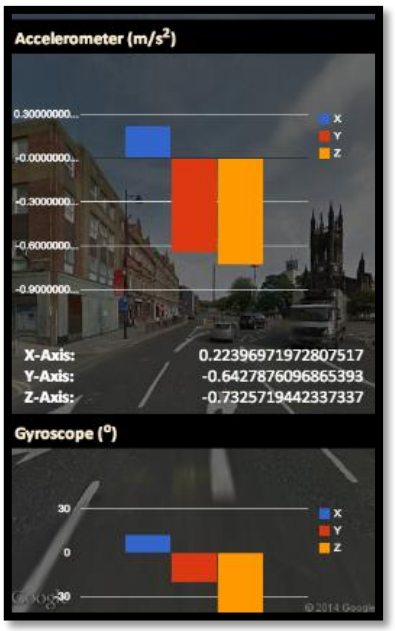
- "Application was too slow to load and overall looked very drab."
- "The Google Street View as the background was quite enticing but a graph or something similar would have been apt to display and compare graph values in real-time."
- "A nice, dark colour scheme with minimal buttons and layout would enhance the application and make it interesting."
- "Overall done well, but the fact that it takes time to load is quite annoying."
- "Good outlook but needs a better colour scheme and something to visualize data."

Alpha version feedback:

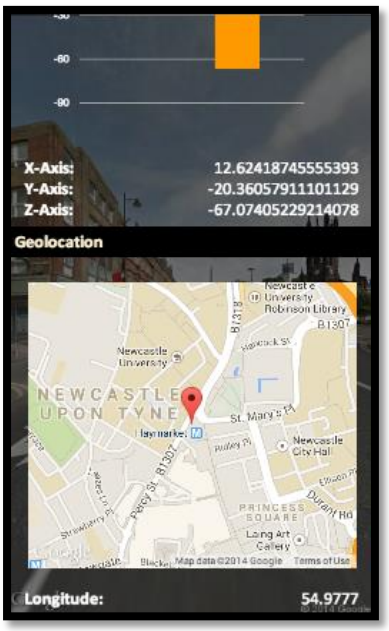
- “Application loads faster, and the new look is quite fresh and interesting.”
- “The new look has enhanced the application. New implementation of Google Street View is very apt and brilliant.”
- “Colour scheme could have been better chosen. But minimal layout is suiting well for the application.”
- “Loading time is still the same.”
- “Visualization is very good, but I would have liked a collective visualization of data.”

8. CONCLUSION

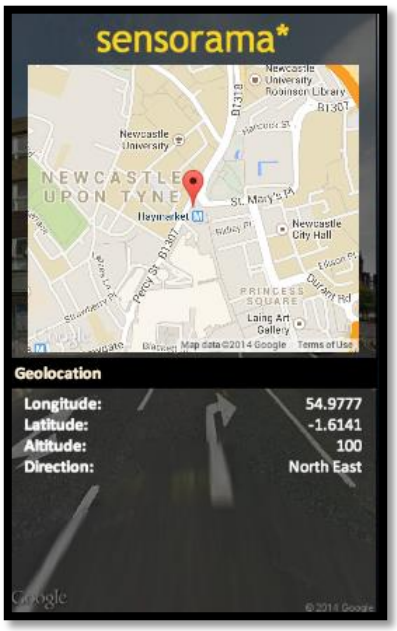
Given below are some screenshots of the app, depicting the final results of this project:



All Sensors – Part 1



All Sensors – Part 2



Geolocation

Gyroscope

The above screenshots are actual screenshots and not representational screenshots. They were taken while the app was running on a real device.

8.1. OVERVIEW

The application managed to achieve most of its objectives and worked perfectly fine after the final Alpha release. There were some objectives that were not achievable due to some restrictions. These restrictions and the achieved and unachieved objectives will be discussed in the following section.

8.2. AIMS AND OBJECTIVES

1. To identify the key sensors used very often in a smartphone
    - The sensors were duly identified as the accelerometer and the geolocation sensors.
    - They were successfully implemented.
2. To identify how the sensor data can be visualised in a simple manner for the understanding of a novice user
    - Line graphs were ruled out. Bar graphs were chosen.
    - Simple visualization technique was implemented.
3. To identify the requirements of the sensor data visualizer
    - Requirements analysed as colour coding, simplicity and real time comparison.
    - Above mentioned requirements implemented.
4. To develop a sensor data visualizer while keeping in mind the “heaviness” of the app
    - App passes the “heaviness” criteria.
    - App deemed light weight through emulator scoring and user feedbacks.
5. To allow the user to record and later review the sensor data
    - Objective unachievable as HTML5 currently has no local storage API support.
    - Objective added as a future improvement as API is under way.
6. To create an interactive multigraph for the sensor data across time for an effective comparison
    - Multigraph could not be implemented as HTML5 and JavaScript are deemed too insecure to allow root access.
    - Multigraph added as a future improvement.
7. To evaluate the effectiveness of the app
    - Overall average score of the app across multiple platforms yielded a 7/10. App was effective but can use improvements.

8.3. SUMMARY

This project was effective in terms of implementing the desired objectives. Some key objectives that could not be implemented due to restrictions with the languages used and HTML5 not supporting proper file system APIs have been considered as future improvements as HTML5 File System APIs are currently being considered by the W3 consortium and will soon be widely usable.

More sensors could not be added as HTML5 does not provide access to all the sensors. Many of them are available to use through Mozilla Developer platform, but the compatibility is limited to only Mozilla browsers and operating systems.

8.4. FUTURE WORK

As is the case with any application, this project can use a lot of future work with regards to adding many key functions that had to be left out due to language restrictions. Some of them are listed below.

8.4.1. IMPLEMENTATION OF MORE SENSORS

Since HTML5 is relatively new to the development scene, access to all the sensors in a smartphone is currently unavailable. This is soon thought to be rectified as the W3 consortium and developers are keen to introduce more sensor support through HTML5. Currently Mozilla Developer Framework is the only one that supports accessing sensors other that the accelerometer and geolocation sensors, but the compatibility is limited to Mozilla products.

8.4.2. BETTER VISUALIZATION WITH FURTHER API SUPPORT

With more and more APIs being considered for HTML5, an overall visualization and smooth flowing data visualization is surely be something that can be committed to this application.

8.4.3. COLLECTION OF SENSOR DATA

With the File System API under consideration with the W3C, the chances of it being passed are very high. Once the API is released officially with full support, sensor data can be collected and visualised in a very interactive manner and user can then compare, assess and review data.

9. REFERENCES

- Intel XDK Online API Reference Library
  http://www.html5dev-software.intel.com/documentation/

- Opportunistic Mobile Sensor Data Collection with SCAR
  By Bence P´asztor, Mirco Musolesi and Cecilia Mascolo
  http://www.cl.cam.ac.uk/~bp296/papers/mass07.pdf

- The Interaction of Data Representation and Data Routing in Sensors
  By Deepak Ganesan
  http://works.bepress.com/deepak_ganesan/1

- HTML5 Article Directory
  http://www.htmlgoodies.com/html5/index.php

- Understanding your HTML5 development options
  https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options

- Cross-Platform Mobile App Software Development in the Curriculum
  By Kyle Lutes
  http://iisit.org/Vol9/IISITv9p115-124Lutes120.pdf

10. APPENDICES

Appendices for the dissertation start from this page forth.

10.1 SOURCE CODES

The source codes for all files involved in the dissertation.

10.1.1. INDEX.HTML

```
<!DOCTYPE html><!--HTML5 doctype-->
<html>
<head>
	<title>Sensorama*</title>
	<meta http-equiv="Content-type" content="text/html; charset=utf-8">
	<meta name="viewport" content="width=device-width, initial-
scale=1.0, maximum-scale=1.0, minimum-scale=1.0, user-scalable=0" />
	<script src="js/jquery-2.1.1.js"></script>
	<link rel="stylesheet" type="text/css" href="style/style.css" />
	<style type="text/css">
		/* Prevent copy paste for all elements except text fields
*/
		*  { -webkit-user-select:none; -webkit-tap-highlight-
color:rgba(255, 255, 255, 0); }
		input, textarea  { -webkit-user-select:text; }
	</style>
	<script src='intelxdk.js'></script>
	<script type="text/javascript">
		/* This code is used to run as soon as Intel activates */
		var onDeviceReady=function(){
		//hide splash screen
		intel.xdk.device.hideSplashScreen();
		};

	document.addEventListener("intel.xdk.device.ready",onDeviceReady,fa
lse);
	</script>

</head>
<body>

<content>

	<a href="index.html">
		<logo>
			sensorama*
		</logo>
	</a>

		<div class="mmenu">
		<p align="center">
			<a href="all.html">
				<img src="images/all.png" class="mmenu" />
			</a>
		<br />
			<a href="acc.html">
				<img src="images/acc.png" class="mmenu" />
			</a>
```

```
                <br />
                        <a href="gyro.html">
                                <img src="images/gyro.png" class="mmenu" />
                        </a>
                <br />
                        <a href="geo.html">
                                <img src="images/maps.png" class="mmenu" />
                        </a>
                <br />
                        <a href="info.html">
                                <img src="images/info.png" class="mmenu" />
                        </a>
                </p>
                </div>

</content>

</body>
</html>
```

10.1.2. LOAD.HTML

```
<!DOCTYPE html><!--HTML5 doctype-->
<html>
<head>
        <title>Sensorama - Load</title>
        <meta http-equiv="Content-type" content="text/html; charset=utf-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0, maximum-scale=1.0, minimum-scale=1.0, user-scalable=0" />
        <script src="js/jquery-2.1.1.js"></script>
        <link rel="stylesheet" type="text/css" href="style/style.css" />
        <style type="text/css">
                /* Prevent copy paste for all elements except text fields
*/
                *  { -webkit-user-select:none; -webkit-tap-highlight-
color:rgba(255, 255, 255, 0); }
                input, textarea  { -webkit-user-select:text; }
        </style>
        <script src='intelxdk.js'></script>
        <script type="text/javascript">
                /* This code is used to run as soon as Intel activates */
                var onDeviceReady=function(){
                //hide splash screen
                intel.xdk.device.hideSplashScreen();
                };

        document.addEventListener("intel.xdk.device.ready",onDeviceReady,fa
lse);
        </script>
</head>
<body>
        <center>
        <img src="images/load.gif" style="display:block; margin-top: 35%;"
/>
        </center>
</body>
</html>
```

10.1.3. INFO.HTML

```
<!DOCTYPE html><!--HTML5 doctype-->
<html>
<head>
	<title>Sensorama*</title>
	<meta http-equiv="Content-type" content="text/html; charset=utf-8">
	<meta name="viewport" content="width=device-width, initial-
scale=1.0, maximum-scale=1.0, minimum-scale=1.0, user-scalable=0" />
	<script src="js/jquery-2.1.1.js"></script>
	<link rel="stylesheet" type="text/css" href="style/style.css" />
	<style type="text/css">
		/* Prevent copy paste for all elements except text fields
*/
		*  { -webkit-user-select:none; -webkit-tap-highlight-
color:rgba(255, 255, 255, 0); }
		input, textarea  { -webkit-user-select:text; }
	</style>
	<script src='intelxdk.js'></script>
	<script type="text/javascript">
		/* This code is used to run as soon as Intel activates */
		var onDeviceReady=function(){
		//hide splash screen
		intel.xdk.device.hideSplashScreen();
		};

	document.addEventListener("intel.xdk.device.ready",onDeviceReady,fa
lse);
	</script>

</head>
<body>

<content>

	<a href="index.html">
		<logo>
			sensorama*
		</logo>
	</a>

	<div class="boundbox" align="justify">
	<p>
		Sensorama as an application is a simple sensor data
visualiser. This application does not
		store data and or display data publicly. Utmost care has
been taken to ensure data is as secure as possible.
	</p>
	<p>
		By using this app, the user agrees the risks involved and
willingly provides this app complete access to
		the geolocation and accelerometer data through this device.
In case of any discrepancy, the developer
		will in no way be held responsible.
	</p>
	<p>
		This app cannot be recreated, sold and/or leased in any way
without contacting the developer at the following
		email address: samrith@outlook.com
	</p>
	</div>
```

```
</content>
</body>
</html>
```

10.1.4. ACC.HTML

```
<!DOCTYPE html><!--HTML5 doctype-->
<html>
<head>
        <title>Sensorama - Accelerometer</title>
        <meta http-equiv="Content-type" content="text/html; charset=utf-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0, maximum-scale=1.0, minimum-scale=1.0, user-scalable=0" />
        <link rel="stylesheet" type="text/css" href="style/style.css" />
        <style type="text/css">
                /* Prevent copy paste for all elements except text fields
*/
                *  { -webkit-user-select:none; -webkit-tap-highlight-
color:rgba(255, 255, 255, 0); }
                input, textarea  { -webkit-user-select:text; }
        </style>
        <script src='intelxdk.js'></script>
         <script type="text/javascript"
src="https://www.google.com/jsapi"></script>
        <script type="text/javascript">

/*jslint browser:true, devel:true, white:true, vars:true, eqeq:true */
/*global intel:false*/

/*
 * This function runs once the page is loaded, but the JavaScript bridge
library is not yet active.
 */

var init = function () {

};

window.addEventListener("load", init, false);

// Prevent Default Scrolling
var preventDefaultScroll = function(event)
{
    // Prevent scrolling on this element
    event.preventDefault();
    window.scroll(0,0);
    return false;
};

window.document.addEventListener("touchmove", preventDefaultScroll, false);

//Options for watch accelaration
var options = { frequency: 1, adjustForRotation: false };

/*
 * Device Ready Code
 * This event handler is fired once the JavaScript bridge library is ready
 */
function onDeviceReady()
{
```

```
    //lock orientation
    intel.xdk.device.setRotateOrientation("portrait");
    intel.xdk.device.setAutoRotate(false);

    //manage power
    intel.xdk.device.managePower(true,false);

    //hide splash screen
    intel.xdk.device.hideSplashScreen();

    //Watch acceleration
    try {
        intel.xdk.accelerometer.watchAcceleration(successAcc, options);
    }
    catch (error) {
        alert("Accelerometer error: \n" + error.name + ": " +
error.message);
        }
        try {
        intel.xdk.geolocation.watchPosition(setFrame);
    }
    catch (error) {
        alert("GPS error: \n" + error.name + ": " + error.message);
    }
}

document.addEventListener("intel.xdk.device.ready",onDeviceReady,false);

//function that modifies the acceleration
function successAcc(acceleration)
{
        var acc;

    acc = document.getElementById("accx");
    acc.innerHTML = acceleration.x;

        acc = document.getElementById("accy");
        acc.innerHTML = acceleration.y;

        acc = document.getElementById("accz");
        acc.innerHTML = acceleration.z;

        drawChart(acceleration);
}

google.load("visualization", "1", {packages:["corechart"]});
google.setOnLoadCallback(drawChart);

function drawChart(acceleration) {

        var ax = acceleration.x;
        var ay = acceleration.y;
        var az = acceleration.z;

        var data = google.visualization.arrayToDataTable([['T', 'X', 'Y',
'Z'], [acceleration.getTime, ax, ay, az]]);

        var options = {
                backgroundColor: 'transparent',
                legend: {postion: 'top',
                                textStyle: { color: '#fff' }
```

```
                                },
                hAxis: { textStyle: { color: '#fff' } },
                vAxis: { textStyle: { color: '#fff' } }
    }

        var chart = new
google.visualization.ColumnChart(document.getElementById('chart_div'));
        chart.draw(data, options);
}

function setFrame(position) {

    var element = document.getElementById('mapframe');
    element.src = "http://maps.googleapis.com/maps/api/streetview?size=" +
element.clientWidth + "x" + element.clientHeight + "&location=" +
position.coords.latitude + "," + position.coords.longitude + "&heading=" +
position.coords.heading + "&pitch=-0&sensor=true";

}
</script>
</head>
<body>

<content>

<a href="index.html">
                <logo>
                        sensorama*
                </logo>
        </a>

<div id="chart_div" style="height: 300px; width: 100%;"></div>
        <h1>Accelerometer (m/s<sup>2</sup>)</h1>
        <div class="boundbox">
                <b>X-Axis: </b>
                <span id="accx">-</span>
                <br />
                <b>Y-Axis: </b>
                <span id="accy">-</span>
                <br />
                <b>Z-Axis: </b>
                <span id="accz">-</span>
        </div>

</content>

        <iframe src="load.html" id="mapframe">iFrame unsupported</iframe>

</body>
</html>
```

10.1.5. GYRO.HTML

```
<!DOCTYPE html><!--HTML5 doctype-->
<html>
<head>
        <title>Sensorama - Gyroscope</title>
        <meta http-equiv="Content-type" content="text/html; charset=utf-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0, maximum-scale=1.0, minimum-scale=1.0, user-scalable=0" />
```

```
        <script src="js/jquery-2.1.1.js"></script>
        <script src="js/main.js"></script>
        <link rel="stylesheet" type="text/css" href="style/style.css" />
        <style type="text/css">
                /* Prevent copy paste for all elements except text fields
*/
                *  { -webkit-user-select:none; -webkit-tap-highlight-
color:rgba(255, 255, 255, 0); }
                input, textarea  { -webkit-user-select:text; }
        </style>
        <script src='intelxdk.js'></script>
        <script type="text/javascript">
                /* This code is used to run as soon as Intel activates */
                var onDeviceReady=function(){
                //hide splash screen
                intel.xdk.device.hideSplashScreen();
                };

        document.addEventListener("intel.xdk.device.ready",onDeviceReady,fa
lse);
        </script>
         <script type="text/javascript"
src="https://www.google.com/jsapi"></script>
        <script type="text/javascript">

/*jslint browser:true, devel:true, white:true, vars:true, eqeq:true */
/*global intel:false*/

/*
 * This function runs once the page is loaded, but the JavaScript bridge
library is not yet active.
 */

var init = function () {

};

window.addEventListener("load", init, false);

// Prevent Default Scrolling
var preventDefaultScroll = function(event)
{
    // Prevent scrolling on this element
    event.preventDefault();
    window.scroll(0,0);
    return false;
};

window.document.addEventListener("touchmove", preventDefaultScroll, false);

//Options for watch accelaration
var options = { frequency: 1, adjustForRotation: false };

/*
 * Device Ready Code
 * This event handler is fired once the JavaScript bridge library is ready
 */
function onDeviceReady()
{
    //lock orientation
    intel.xdk.device.setRotateOrientation("portrait");
```

```
    intel.xdk.device.setAutoRotate(false);

    //manage power
    intel.xdk.device.managePower(true,false);

    //hide splash screen
    intel.xdk.device.hideSplashScreen();

    //Watch acceleration
    try {
        intel.xdk.accelerometer.watchAcceleration(successAcc);
    }
    catch (error) {
        alert("Accelerometer error: \n" + error.name + ": " +
error.message);
        }
        try {
        intel.xdk.geolocation.watchPosition(setFrame);
    }
    catch (error) {
        alert("GPS error: \n" + error.name + ": " + error.message);
    }
}

document.addEventListener("intel.xdk.device.ready",onDeviceReady,false);

//function that modifies the acceleration
function successAcc(acceleration)
{
    var element;

        var ax = acceleration.x;
    var ay = acceleration.y;
    var az = acceleration.z;

    var xAngle = Math.atan( ax / (Math.sqrt(ay^2 + az)));
    var yAngle = Math.atan( ay / (Math.sqrt(ax^2 + az^2)));
    var zAngle = Math.atan( Math.sqrt(ax^2 + ay^2) / az);

    xAngle *= 180.00;   yAngle *= 180.00;   zAngle *= 180.00;
    xAngle /= 3.141592; yAngle /= 3.141592; zAngle /= 3.141592;

    element = document.getElementById('gyrox');
    element.innerHTML = xAngle;

    element = document.getElementById('gyroy');
        element.innerHTML = yAngle;

    element = document.getElementById('gyroz');
        element.innerHTML = zAngle;

        drawChart(acceleration);
}
google.load("visualization", "1", {packages:["corechart"]});
google.setOnLoadCallback(drawChart);

function drawChart(acceleration) {

        var ax = acceleration.x;
        var ay = acceleration.y;
        var az = acceleration.z;
```

```
        var xAngle = Math.atan( ax / (Math.sqrt(ay^2 + az)));
    var yAngle = Math.atan( ay / (Math.sqrt(ax^2 + az^2)));
    var zAngle = Math.atan( Math.sqrt(ax^2 + ay^2) / az);

    xAngle *= 180.00;   yAngle *= 180.00;   zAngle *= 180.00;
    xAngle /= 3.141592; yAngle /= 3.141592; zAngle /= 3.141592;

        var data = google.visualization.arrayToDataTable([['T', 'X', 'Y',
'Z'], [acceleration.getTime, xAngle, yAngle, zAngle]]);

        var options = {
                backgroundColor: 'transparent',
                legend: {postion: 'bottom',
                                textStyle: { color: '#fff' }
                               },
                hAxis: { textStyle: { color: '#fff' } },
                vAxis: { textStyle: { color: '#fff' } }
    }

        var chart = new
google.visualization.ColumnChart(document.getElementById('chart_div'));
        chart.draw(data, options);
}

function setFrame(position) {

    var element = document.getElementById('mapframe');
    element.src = "http://maps.googleapis.com/maps/api/streetview?size=" +
element.clientWidth + "x" + element.clientHeight + "&location=" +
position.coords.latitude + "," + position.coords.longitude + "&heading=" +
position.coords.heading + "&pitch=-0&sensor=true";

}
</script>
</head>
<body>
<content>

        <a href="index.html">
                <logo>
                        sensorama*
                </logo>
        </a>

        <div id="chart_div" style="height: 300px; width: 100%;"></div>
        <h1>Gyroscope (<sup>o</sup>)</h1>
        <div class="boundbox">
                <b>X-Axis: </b>
                <span id="gyrox">-</span>
                <br />
                <b>Y-Axis: </b>
                <span id="gyroy">-</span>
                <br />
                <b>Z-Axis: </b>
                <span id="gyroz">-</span>
        </div>

</content>

<iframe src="load.html" id="mapframe">iFrame unsupported</iframe>
```

```
</body>
</html>
```

10.1.6. GEO.HTML

```
<!DOCTYPE html><!--HTML5 doctype-->
<html>
<head>
        <title>Sensorama - Geolocation</title>
        <meta http-equiv="Content-type" content="text/html; charset=utf-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0, maximum-scale=1.0, minimum-scale=1.0, user-scalable=0" />
        <script src="js/jquery-2.1.1.js"></script>
        <script src="js/main.js"></script>
        <link rel="stylesheet" type="text/css" href="style/style.css" />
        <style type="text/css">
                /* Prevent copy paste for all elements except text fields
*/
                *  { -webkit-user-select:none; -webkit-tap-highlight-
color:rgba(255, 255, 255, 0); }
                input, textarea  { -webkit-user-select:text; }
        </style>
        <script src='intelxdk.js'></script>
        <script type="text/javascript">
                /* This code is used to run as soon as Intel activates */
                var onDeviceReady=function(){
                //hide splash screen
                intel.xdk.device.hideSplashScreen();
                };

        document.addEventListener("intel.xdk.device.ready",onDeviceReady,fa
lse);
        </script>
         <script
src="//maps.googleapis.com/maps/api/js?v=3.exp&sensor=true"></script>
        <script type="text/javascript">

/*jslint browser:true, devel:true, white:true, vars:true, eqeq:true */
/*global intel:false*/

/*
 * This function runs once the page is loaded, but the JavaScript bridge
library is not yet active.
 */

var init = function () {

};

window.addEventListener("load", init, false);

// Prevent Default Scrolling
var preventDefaultScroll = function(event)
{
    // Prevent scrolling on this element
    event.preventDefault();
    window.scroll(0,0);
    return false;
};
```

```
window.document.addEventListener("touchmove", preventDefaultScroll, false);

//Options for watch accelaration
var options = { frequency: 1, adjustForRotation: false };

/*
 * Device Ready Code
 * This event handler is fired once the JavaScript bridge library is ready
 */
function onDeviceReady()
{
    //lock orientation
    intel.xdk.device.setRotateOrientation("portrait");
    intel.xdk.device.setAutoRotate(false);

    //manage power
    intel.xdk.device.managePower(true,false);

    //hide splash screen
    intel.xdk.device.hideSplashScreen();


     try {
        intel.xdk.geolocation.watchPosition(successGeo);
    }
    catch (error) {
        alert("GPS error: \n" + error.name + ": " + error.message);
    }
}

function successGeo(position)
{
    var element;
    var direction;

    var compass = document.getElementById('compass');

    if(position.coords.heading === 0 || position.coords.heading === 360)
        direction = "North";
    if(position.coords.heading> 0 && position.coords.heading < 90)
        direction = "North East";
    if(position.coords.heading === 90)
        direction = "East";
    if(position.coords.heading>90 && position.coords.heading < 180)
        direction = "South East";
    if(position.coords.heading === 180)
        direction = "South";
    if(position.coords.heading> 180 && position.coords.heading < 270)
        direction = "South West";
    if(position.coords.heading === 270)
        direction = "West";
    if(position.coords.heading>270 && position.coords.heading<360)
        direction = "North West";

            element = document.getElementById('geo1');
        element.innerHTML = position.coords.latitude;

            element = document.getElementById('geo2');
        element.innerHTML = position.coords.longitude;
```

```
                element = document.getElementById('geo3');
        element.innerHTML = position.coords.altitude;

        compass.innerHTML = direction;

        setFrame(position);

        (function() {

                if(!!navigator.geolocation) {

                        var map;

                        var mapOptions = {
                                zoom: 15,
                                mapTypeId: google.maps.MapTypeId.ROADMAP,
                                draggable: false,
                                zoomControl: false,
                                scrollWheel: false,
                                disableDoubleClickZoom: true,
                                disableDefaultUI: true
                        };

                        map = new
google.maps.Map(document.getElementById('google_canvas'), mapOptions);


        navigator.geolocation.getCurrentPosition(function(position) {

                                var geolocate = new
google.maps.LatLng(position.coords.latitude, position.coords.longitude);

                                var marker = new google.maps.Marker({
                                        map: map,
                                        position: geolocate,
                                });

                                map.setCenter(geolocate);

                        });

                } else {
                        document.getElementById('google_canvas').innerHTML =
'No Geolocation Support.';
                }

            })();
}


function setFrame(position) {

    var element = document.getElementById('mapframe');
    element.src = "http://maps.googleapis.com/maps/api/streetview?size=" +
element.clientWidth + "x" + element.clientHeight + "&location=" +
position.coords.latitude + "," + position.coords.longitude + "&heading=" +
position.coords.heading + "&pitch=-0&sensor=true";

}
</script>
</head>
```

```
<body>

<content>

        <a href="index.html">
                <logo>
                        sensorama*
                </logo>
        </a>

        <center><div id="google_canvas" style="width:90%;
height:300px;"></div></center>
        <h1>Geolocation</h1>
        <div class="boundbox">
                <b>Longitude: </b>
                <span id="geo1">-</span>
                <br />
                <b>Latitude: </b>
                <span id="geo2">-</span>
                <br />
                <b>Altitude: </b>
                <span id="geo3">-</span>
                <br />
                <b>Direction: </b>
                <span id="compass">-</span>
        </div>

</content>

<iframe src="load.html" id="mapframe">iFrame unsupported</iframe>

</body>
</html>
```

10.1.7 ALL.HTML

```
<!DOCTYPE html><!--HTML5 doctype-->
<html>
<head>
        <title>Sensorama - All Sensors</title>
        <meta http-equiv="Content-type" content="text/html; charset=utf-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0, maximum-scale=1.0, minimum-scale=1.0, user-scalable=0" />
        <script src="js/jquery-2.1.1.js"></script>
        <script src="js/main.js"></script>
        <link rel="stylesheet" type="text/css" href="style/style.css" />
        <style type="text/css">
                /* Prevent copy paste for all elements except text fields
*/
                *  { -webkit-user-select:none; -webkit-tap-highlight-
color:rgba(255, 255, 255, 0); }
                input, textarea  { -webkit-user-select:text; }
        </style>
        <script src='intelxdk.js'></script>
        <script type="text/javascript">
                /* This code is used to run as soon as Intel activates */
                var onDeviceReady=function(){
                //hide splash screen
                intel.xdk.device.hideSplashScreen();
                };
```

```
        document.addEventListener("intel.xdk.device.ready",onDeviceReady,fa
lse);
        </script>
         <script type="text/javascript"
src="https://www.google.com/jsapi"></script>
         <script
src="//maps.googleapis.com/maps/api/js?v=3.exp&sensor=true"></script>
        <script type="text/javascript">

/*jslint browser:true, devel:true, white:true, vars:true, eqeq:true */
/*global intel:false*/

/*
 * This function runs once the page is loaded, but the JavaScript bridge
library is not yet active.
 */

var init = function () {

};

window.addEventListener("load", init, false);

// Prevent Default Scrolling
var preventDefaultScroll = function(event)
{
    // Prevent scrolling on this element
    event.preventDefault();
    window.scroll(0,0);
    return false;
};

window.document.addEventListener("touchmove", preventDefaultScroll, false);

//Options for watch accelaration
var options = { frequency: 1, adjustForRotation: false };

/*
 * Device Ready Code
 * This event handler is fired once the JavaScript bridge library is ready
 */
function onDeviceReady()
{
    //lock orientation
    intel.xdk.device.setRotateOrientation("portrait");
    intel.xdk.device.setAutoRotate(false);

    //manage power
    intel.xdk.device.managePower(true,false);

    //hide splash screen
    intel.xdk.device.hideSplashScreen();

    //Watch acceleration
    try {
        intel.xdk.accelerometer.watchAcceleration(successAcc);
    }
    catch (error) {
        alert("Accelerometer error: \n" + error.name + ": " +
error.message);
```

```
        }
        try {
        intel.xdk.geolocation.watchPosition(successGeo);
    }
    catch (error) {
        alert("GPS error: \n" + error.name + ": " + error.message);
    }
}

document.addEventListener("intel.xdk.device.ready",onDeviceReady,false);

//function that modifies the acceleration
function successAcc(acceleration)
{
        var acc;

    acc = document.getElementById("accx");
    acc.innerHTML = acceleration.x;

        acc = document.getElementById("accy");
        acc.innerHTML = acceleration.y;

        acc = document.getElementById("accz");
        acc.innerHTML = acceleration.z;

    var element;

        var ax = acceleration.x;
    var ay = acceleration.y;
    var az = acceleration.z;

    var xAngle = Math.atan( ax / (Math.sqrt(ay^2 + az)));
    var yAngle = Math.atan( ay / (Math.sqrt(ax^2 + az^2)));
    var zAngle = Math.atan( Math.sqrt(ax^2 + ay^2) / az);

    xAngle *= 180.00;   yAngle *= 180.00;   zAngle *= 180.00;
    xAngle /= 3.141592; yAngle /= 3.141592; zAngle /= 3.141592;

    element = document.getElementById('gyrox');
    element.innerHTML = xAngle;

    element = document.getElementById('gyroy');
        element.innerHTML = yAngle;

    element = document.getElementById('gyroz');
        element.innerHTML = zAngle;

        drawChart(acceleration);
}

google.load("visualization", "1", {packages:["corechart"]});
google.setOnLoadCallback(drawChart);

function drawChart(acceleration) {

        var ax = acceleration.x;
        var ay = acceleration.y;
        var az = acceleration.z;

        var xAngle = Math.atan( ax / (Math.sqrt(ay^2 + az)));
    var yAngle = Math.atan( ay / (Math.sqrt(ax^2 + az^2)));
```

```
    var zAngle = Math.atan( Math.sqrt(ax^2 + ay^2) / az);

    xAngle *= 180.00;   yAngle *= 180.00;   zAngle *= 180.00;
    xAngle /= 3.141592; yAngle /= 3.141592; zAngle /= 3.141592;


        var data1 = google.visualization.arrayToDataTable(
                                [['T', 'X', 'Y', 'Z'],
                                [acceleration.getTime, ax, ay, az]]);

        var data2 = google.visualization.arrayToDataTable(
                                [['T', 'X', 'Y', 'Z'],
                                [acceleration.getTime, xAngle, yAngle,
zAngle]]);

        var options = {
                backgroundColor: 'transparent',
                legend: {postion: 'bottom',
                                 textStyle: { color: '#fff' }
                                },
                hAxis: { textStyle: { color: '#fff' } },
                vAxis: { textStyle: { color: '#fff' } }
    }

        var chart1 = new
google.visualization.ColumnChart(document.getElementById('chart_div1'));
        chart1.draw(data1, options);

    var chart2 = new
google.visualization.ColumnChart(document.getElementById('chart_div2'));
        chart2.draw(data2, options);
}

function successGeo(position) {

    var element;
    var direction;

    var compass = document.getElementById('compass');

    if(position.coords.heading === 0 || position.coords.heading === 360)
        direction = "North";
    if(position.coords.heading> 0 && position.coords.heading < 90)
        direction = "North East";
    if(position.coords.heading === 90)
        direction = "East";
    if(position.coords.heading>90 && position.coords.heading < 180)
        direction = "South East";
    if(position.coords.heading === 180)
        direction = "South";
    if(position.coords.heading> 180 && position.coords.heading < 270)
        direction = "South West";
    if(position.coords.heading === 270)
        direction = "West";
    if(position.coords.heading>270 && position.coords.heading<360)
        direction = "North West";

                element = document.getElementById('geo1');
        element.innerHTML = position.coords.latitude;

                element = document.getElementById('geo2');
```

```
        element.innerHTML = position.coords.longitude;

                element = document.getElementById('geo3');
        element.innerHTML = position.coords.altitude;

        compass.innerHTML = direction;

    var element = document.getElementById('mapframe');
    element.src = "http://maps.googleapis.com/maps/api/streetview?size=" +
element.clientWidth + "x" + element.clientHeight + "&location=" +
position.coords.latitude + "," + position.coords.longitude + "&heading=" +
position.coords.heading + "&pitch=-0&sensor=true";

    (function() {

                if(!!navigator.geolocation) {

                        var map;

                        var mapOptions = {
                                zoom: 15,
                                mapTypeId: google.maps.MapTypeId.ROADMAP,
                                draggable: false,
                                zoomControl: false,
                                scrollWheel: false,
                                disableDoubleClickZoom: true,
                                disableDefaultUI: true
                        };

                        map = new
google.maps.Map(document.getElementById('google_canvas'), mapOptions);


        navigator.geolocation.getCurrentPosition(function(position) {

                                var geolocate = new
google.maps.LatLng(position.coords.latitude, position.coords.longitude);

                                var marker = new google.maps.Marker({
                                        map: map,
                                        position: geolocate,
                                });

                                map.setCenter(geolocate);

                        });

                } else {
                        document.getElementById('google_canvas').innerHTML =
'No Geolocation Support.';
                }

            })();
}

</script>
</head>
<body>
<content>

        <a href="index.html">
```

```
                <logo>
                        sensorama*
                </logo>
        </a>

        <h1>Accelerometer (m/s<sup>2</sup>)</h1>
        <div id="chart_div1" style="height: 300px; width: 100%;"></div>
        <div class="boundbox">
                <b>X-Axis: </b>
                <span id="accx">-</span>
                <br />
                <b>Y-Axis: </b>
                <span id="accy">-</span>
                <br />
                <b>Z-Axis: </b>
                <span id="accz">-</span>
        </div>

        <h1>Gyroscope (<sup>o</sup>)</h1>
        <div id="chart_div2" style="height: 300px; width: 100%;"></div>
        <div class="boundbox">
                <b>X-Axis: </b>
                <span id="gyrox">-</span>
                <br />
                <b>Y-Axis: </b>
                <span id="gyroy">-</span>
                <br />
                <b>Z-Axis: </b>
                <span id="gyroz">-</span>
        </div>

        <h1>Geolocation</h1>
        <br />
        <center><div id="google_canvas" style="width:90%;
height:300px;"></div></center>
        <br />
        <div class="boundbox">
                <b>Longitude: </b>
                <span id="geo1">-</span>
                <br />
                <b>Latitude: </b>
                <span id="geo2">-</span>
                <br />
                <b>Altitude: </b>
                <span id="geo3">-</span>
                <br />
                <b>Direction: </b>
                <span id="compass">-</span>
        </div>
        <br /><br />

</content>

<iframe src="load.html" id="mapframe">iFrame unsupported</iframe>

</body>
</html>
```

10.2. IMAGE APPENDIX

The links for the sources of images used are given below.

- Figure 1.1 - http://developer.getpebble.com/2/guides/accel.png
- Figure 3.1.1.1 - http://www.silverlightshow.net/Storage/Users/MisterGoodcat/Accelerometer-axes_2.png
- Figure 3.1.2.1 - http://www.analog.com/library/analogdialogue/archives/37-03/Gyro-01.gif
- Figure 3.1.3.1 - http://www.tik.ee.ethz.ch/~beutel/pics/rel_pos_triangulation.jpg
- Figure 3.2.1 - http://www.icapps.com/wp-content/uploads/2012/06/04-HTML5-vs-Native.008.png
- Figure 3.4.1.1 and Figure 3.4.1.2 - iStore and Google Play Store