Enabling the Virtual Phones to Remotely Sense the Real Phones in Real-Time

~ A Sensor Emulation initiative for virtualized Android-x86 ~

Raghavan Santhanam

Submitted in partial fulfillment of the
requirements for the degree
of Master of Science in Computer Science
in the School of Engineering and Applied Science

2014

ABSTRACT

Enabling the Virtual Phones to Remotely Sense the Real Phones in Real-Time

~ A Sensor Emulation initiative for virtualized Android-x86 ~

Smartphones nowadays have the ground-breaking features that were only a figment of one's imagination. For the ever-demanding cellphone users, the exhaustive list of features that a smartphone supports just keeps getting more exhaustive with time. These features aid one's personal and professional uses as well. Extrapolating into the future the features of a present-day smartphone, the lives of us humans using smartphones are going to be unimaginably agile.

With the above said emphasis on the current and future potential of a smartphone, the ability to virtualize smartphones with all their real-world features into a virtual platform, is a boon for those who want to rigorously experiment and customize the virtualized smartphone hardware without spending an extra penny. Once virtualizable independently on a larger scale, the idea of virtualized smartphones with all the virtualized pieces of hardware takes an interesting turn with the sensors being virtualized in a way that's closer to the real-world behavior.

When accessible remotely with the real-time responsiveness, the above mentioned real-world behavior will be a real dealmaker in many real-world systems, namely, the life-saving systems like the ones that instantaneously get alerts about harmful magnetic radiations in the deep mining areas, etc. And these life-saving systems would be installed on a large scale on the desktops or large servers as virtualized smartphones having the added support of virtualized sensors which remotely fetch the real hardware sensor readings from a real smartphone in real-time. Based on these readings the lives working in the affected areas can be alerted and thus saved by the people who are operating the at the desktops or large servers hosting the virtualized smartphones.

In addition, the direct and one of the biggest advantages of such a real hardware sensor driven Sensor Emulation in an emulated Android(-x86) environment is that the Android applications that use sensors can now run on the emulator and act under the influence of real hardware sensors' due to the emulated sensors.

The current work of Sensor Emulation is quite unique when compared to the existing and past sensor-related works. The uniqueness comes from the full-fledged sensor

emulation in a virtualized smartphone environment as opposed to building some sophisticated physical systems that usually aggregate the sensor readings from the real hardware sensors, might be in a remote manner and in real-time. For example, wireless sensor networks based remote-sensing systems that install real hardware sensors in remote places and have the sensor readings from all those sensors at a centralized server or something similar, for the necessary real-time or offline analysis. In these systems, apart from collecting mere real hardware sensor readings into a centralized entity, nothing more is being achieved unlike in the current work of Sensor Emulation wherein the emulated sensors behave exactly like the remote real hardware sensors. The emulated sensors can be calibrated, speeded up or slowed down(in terms of their sampling frequency), influence the sensor-based application running inside the virtualized smartphone environment exactly as the real hardware sensors of a real phone would do to the sensor-based application running in that real phone. In essence, the current work is more about generalizing the sensors with all its real-world characteristics as far as possible in a virtualized platform than just a framework to send and receive sensor readings over the network between the real and virtual phones.

Realizing the useful advantages of Sensor Emulation which is about adding virtualized sensors support to emulated environments, the current work emulates a total of ten sensors present in the real smartphone, Samsung Nexus S, an Android device. Virtual phones run Android-x86 while real phones run Android. The real reason behind choosing Android-x86 for virtual phone is that x86-based Android devices are feature-rich over ARM based ones, for example a full-fledged x86 desktop or a tablet has more features than a relatively small smartphone. Out of the ten, five are real sensors and the rest are virtual or synthetic ones. The real ones are Accelerometer, Magnetometer, Light, Proximity, and Gyroscope whereas the virtual ones are Corrected Gyroscope, Linear Acceleration, Gravity, Orientation, and Rotation Vector. The emulated Android-x86 is of Android release version Jelly Bean 4.3.1 which differs only very slightly in terms of bug fixes from Android Jelly Bean 4.3 running on the real smartphone.

One of the noteworthy aspects of the Sensor Emulation accomplished is being demand-less - exactly the same sensor-based Android applications will be able to use the sensors on the real and virtual phones, with absolutely no difference in terms of their sensor-based behavior.

The emulation’s core idea is the socket-communication between two modules of Hardware Abstraction Layer(HAL) which is driver-agnostic, remotely over a wireless network in real-time. Apart from a Paired real-device scenario from which the real hardware sensor readings are fetched, the Sensor Emulation also is compatible with a Remote Server Scenario wherein the artificially generated sensor readings are fetched from a remote server. Due to the Sensor Emulation having been built on mere

end-to-end socket-communication, it's logical and obvious to see that the real and virtual phones can run different Android(-x86) releases with no real impact on the Sensor Emulation being accomplished.

Sensor Emulation once completed was evaluated for each of the emulated sensors using applications from Android Market as well as Amazon Appstore. The applications category include both the basic sensor-test applications that show raw sensor readings, as well as the advanced 3D sensor-driven games which are emulator compatible, especially in terms of the graphics. The evaluations proved the current work of Sensor Emulation to be ***generic***, ***efficient***, ***robust***, ***fast***, ***accurate***, and ***real***.

As of this writing i.e., January 2014, the current work of Sensor Emulation is the sole system-level sensor virtualization work that embraces remoteness in real-time for the emulated Android-x86 systems. It is important to note that though the current work is targeted for Android-x86, the code written for the current work makes no assumptions about underlying platform to be an x86 one. Hence, the work is also logically seen as compatible with ARM based emulated Android environment though not actually tested.

Contents

Contents i

List of Figures ii

List of Tables iii

Acknowledgments iv

1 Introduction 1

2 Motivation and Use Cases 3

3 Requirements 7

4 Design Comparisons 8

5 Android-x86. Sensors and HAL 12

6 Sensor Emulation Software Architecture 25

7 Design Considerations 29

8 Demo 33

9 Implementation 35

10 Evaluation 39

11 Challenges 143

12 Usability Factor 152

13 Related Works 154

14 Conclusion and Future Work 167

15 Bibliography 170

LIST OF FIGURES

4.1 Android Low-Level System Architecture 8

6 Sensor Emulation Software Architecture 25

8.1.1 Snapshot 1 - Emulated Accelerometer in action 33

8.1.2 Snapshot 2 - Emulated Gyroscope in action 33

8.2.1 Snapshot 3 - All 10 sensors in action - remote server scenario . . 34

10.1 Pictorial comparison of the average time delays applicable for the sensor readings in reaching the Virtual Android Device(VAD) from the Real Android Device(RAD) 48

LIST OF TABLES

10.1 Numerical comparison of the average time delays applicable for the sensor readings in reaching the Virtual Android Device(VAD) from the Real Android Device(RAD) 50

Acknowledgments

My thesis work is dedicated to all of my lovely family who have supported throughout my studies from my childhood -- my great parents, **Mrs. Mythili Santhanam** and **Mr. Santhanam Raghavachar**, my kind and helping elder brother **Mr. Venugopal Santhanam**, my caring father's sister, Late **Ramaa S K**, and my affectionate late grandmother **Kamalamma R**. I weigh all of them to be equally important regardless of the order in which their names appear here and I am deeply indebted to all of them as without their support I wouldn't have been what I'm now.

I sincerely thank Songchun Fan, PhD student at Duke University, NC for having the patience in reviewing my updates at every intermediate stage of my thesis work in suggesting what's important and what's not. Appreciate it wholeheartedly.

I want to thank Prof. Jason Nieh, CS Dept, Columbia University, NYC for his valuable suggestions on my thesis work to bring my thesis into a good shape with considerable importance for defending my thesis. It's been nice knowing him since my first semester of my Master's here at Columbia having attended his Operating Systems-I in fall 2012 as my ***first day, first class*** of my Master's here at Columbia!

I want to thank Prof. Luca Carloni, CS Dept, Columbia University, NYC for accepting to serve my thesis defense review committee. It's been great knowing his kindness in approaching students who are willing to interact with him anywhere, anytime, especially even after the course being taught and attended is over! I took his interesting course Computer Architecture with him during fall 2012.

I have always been extremely excited by looking at the passion of Prof. Roxana Geambasu for Distributed Systems and the relevant technology. I admire her energy while teaching the courses which is so contagious! It was really thought-provoking to attend and listen to her Distributed Systems course in my first semester during fall 2012.

Last but not the least, I want to thank "Time-The Almighty".

"The Time has to speak and it will."

Chapter 1
Introduction

Being in 2014, the smartphone industry has almost captivated us humans with its astonishing capabilities. The capabilities that were once seen only as science fiction are now realities. The capabilities include but not limited to sensing heat of a thermal body, atmospheric pressure in a submarine, change in ambience for an auto-brightness light, speed a vehicle, spatial orientation of a stationary object, gravity's influence on an object in space, magnetic radiation in a mining area, geographical position in remote parts of our earth, and a lot more. In recent times, these capabilities have become an inseparable part of our lives due to their limitless usefulness in our day-to-day activities.

Smartphone Virtualization is an intriguing technology with its capability to host different platforms on the same piece of hardware with no additional cost - smartphone customization at its best. The ability to run such virtualized smartphones on machines such as desktops or large servers leverages the importance of smartphone virtualization. The leverage is in the form of the opportunity to run hundreds of such virtualized smartphones independently at the same time on the same machine. And when the sensing capabilities of a real smartphone can be fully experienced in such a vastly scaled up virtualized environment, the sensor-based smartphone applications using the virtualized sensors of the virtualized smartphones, will find the door to exploit the strengths of smartphone virtualization, wide open.

Remote-sensing capability over the environmental changes has been the epitome of many active researches in the areas of Space exploration, Oceanography, etc. When the remote-sensing capability is integral to smartphone virtualization, the resulting opportunities are infinite. In other words, adding Sensor Emulation to the virtualized smartphones that can be controlled in a remote manner, will be a great value addition to the smartphone virtualization technology, with those virtualized smartphones being powered by desktops or large servers as mentioned previously. This value addition will be a great feat when it's achievable in real-time. Thus, enabling virtual(virtualized/emulated) phones(smartphones) to remotely sense physical(real) phones(smartphones) in real-time which is nothing but Sensor Emulation, is noteworthy.

Accomplishing the Sensor Emulation in a virtualized smartphone environment emphasizing the aspects of remoteness and real-timeness in a full-fledged manner is quite different from building sophisticated physical systems for real-time

remote-sensing. The difference lies in the very notion of emulation of sensor with all the real-world characteristics. This implies that apart from being able to collect the raw sensor data remotely in real-time from the real hardware sensors, the emulated sensors can be operated just like real hardware sensors, in the virtualized smartphone environment. As a result, the emulated sensors can be calibrated, altered for their sampling frequency of sensor data, and so on. If need be, the real hardware sensors driving can be operated via the emulated sensors by having two-way communication between the real and emulated sensors. All these will be absent in the usual remote-sensing systems whose aim is to just accumulate data in real-time from the deployed real hardware sensors in the designated remote places, and analyze them live or offline. As a matter of fact, building hundreds of sophisticated remote-sensing physical systems is way expensive than powering hundreds of, even thousands of virtualized smartphones on a single or a minimal set of resourceful desktops or large servers, equipped with emulated sensors driven remotely by real hardware sensors in real-time. Thus, the current work of Sensor Emulation in virtualized smartphone environment stands out among all the sensor-based works(listed in related works Chapter) as of January 2014.

The ambitious task of Sensor Emulation, that is enabling virtual phones to remotely sense physical phones in real-time demands considerable amount of work and is quite challenging as well. When the time available to accomplish it is relatively less, the challenge gets even better. Reason being, the software shipped with the smartphones would not have been be usually designed for the purpose of remote-sensing at all, but only for localized usage by an ordinary cellphone user. The considerable work will be in the form of including features into the software of the virtual and physical phones so that virtual phones powered by desktops or large servers, can do the remote-sensing using the physical phones. The included features must be ***robust***, ***fast***, ***accurate***, ***generic***, ***efficient***, and ***real*** - a challenge that the current work of Sensor Emulation has accomplished in a tight schedule of three months from Sep-Nov, 2013.

As far as the current work is concerned, the real phones will be the ARM based Android smartphones and the virtual phones will be running Android-x86. The reason for choosing Android-x86 for virtual phones is that once accomplished the Sensor Emulation can be used in the virtualized Android-x86 based real phones(devices such as tablets, gadgets, etc) which are feature-rich over the ARM based real Android smartphones when full-fledged user layout of desktops and tablets are considered.

Chapter 2
Motivation and use cases

This chapter discusses about the real driving forces behind the current work of Sensor Emulation and the resulting use cases of the Sensor Emulation accomplished.

2.1 Motivation

The topic of sensors has been widely dealt with in number of research papers with respect to number of real-world scenarios.

1. One of Android tools project focuses on achieving Sensor Emulation in a wired(USB) manner with some delay but only for two sensors.[10]
2. Wireless sensor networks which focus on building a wireless network involving real hardware sensors deployed at remote geographical locations.[52-61]
3. Healthcare sensor-based products attached to patients' bodies to monitor their health based on the sensor data retrievable onto a centralized system operated by doctors.[44-47]
4. Sensor simulation(and not emulation) models that generate artificial sensor data in various scenarios.[9,11,12]
5. Many smartphone virtualization works talk about isolating the virtual machines running on the same smartphone, in accessing the underlying real hardware sensors in a secure and controlled manner.[19-32]

For more specifics on related works, Chapter 13 on "Related works" can be referred.

After reading all of the related works, it must be evident that none of them have an element of emulation of sensors, the emulation of sensors in a virtualized platform with emphasis on real-timeness and remoteness. The emulation of sensors in a virtualized platform is more about having full-fledged emulated sensors with all of the characteristics that real hardware sensors on a real device will have. This implies that it's not only about real hardware sensor data available on a virtualized platform remotely in real-time but also the possibility of driving the emulated sensors by the real hardware sensors remotely in real-time making the emulation of sensors closer to the real-world environment. This very possibility makes the current work to stand out among all the sensor-based works of every kind. And this very possibility is the major motivation factor for the current work of Sensor Emulation to be accomplished emphasizing on the aspects of real-timeness and remoteness.

The ability to drive emulated sensors in a virtualized platform by real hardware sensors present on a real device has a huge impact with respect to smartphone environment. The impact is so huge that it enables the feature-rich mobile platform to be fully functional even when virtualized, specifically in terms of sensor-based behavior. As a direct consequence of the availability of sensors support in an emulated smartphone environment, each and every sensor-based application developed for real smartphone environment will now be able to run absolutely unchanged on the emulated smartphone environment. This enablement for the sensor-based applications to act under the influence of real hardware sensors data available via emulated sensors in a virtualized environment remotely in real-time, has numerous useful implications which are listed under the "Use cases" section next.

The real benefit of such an ability is multiplied when thousands of such virtualized smartphone environment will be hosted independently on a single large machine like a resourceful desktop or a powerful server capable of running the virtualized smartphones in an isolated manner in the fullest execution speed. This leverage obtained by employing large-scale smartphone virtualization opens up wide range of opportunities in the field of Space exploration, Oceanography, etc which in general demand large-scale remote-sensing to happen in real-time as per the dynamics of the respective environment. This approach to have large-scale smartphone virtualization with emulated sensors driven remotely by real hardware sensors(say that are present on a spacecraft in the case of Space exploration, etc) in real-time is ***cost-effective*** and ***easily scaleable*** as opposed to other usual remote-sensing systems which involve a huge manual and financial investment on the sophisticated "physical" systems which interact with the remotely deployed systems(like a spacecraft, etc). So, apart from just being able to get raw sensor data remotely in real-time with no additional money involved in powering thousands of virtualized smartphones on the same physical machine(only for which there will be an initial investment), the virtualized sensors behave as real hardware sensors in every aspect and thus the emulation of sensors driven by real hardware sensors remotely in real-time offers completeness that is either absent or limited in the systems that collect only raw sensor data, of course remotely in real-time. Thus, this completeness achievable through the current work was also one of the motivation factors initially.

As of January 2014, among the smartphone platforms, Android is the leading mobile operating system[73]. It's also open source which allows the unrestricted study of the internals of a mobile platform which is very essential for any breakthrough research to happen. In addition, the developer base for the Android is the biggest as of January 2014[74]. And with the added benefits of full-fledged desktop and tablet layout, Android-x86[1], the x86 port of Android which can be run on the desktops and tablets with all of the standard Android features and even more, ideally suits and hence it was one of the motivation factors to accomplish the current work of Sensor Emulation in virtualized Android-x86 emphasizing the aspects of real-timeness and remoteness.

On the whole, the importance and benefits of Sensor Emulation in virtualized Android-x86 emphasizing remoteness and real-timeness, together form a big motivation for the current work.

2.2 Use cases

Use cases of the Sensor Emulation are diverse due to the scalability factor enabled by virtualization in running hundreds of thousands of virtualized Android-x86 phones in parallel on the same desktops or large servers. And the sheer diversity is dominated by the real-timeness and remoteness of the sensor readings provided by the emulated sensors.

There are number of notable use cases including but not limited to the below ones.

1. Monitoring the body temperature of all the patients in a hospital sitting in one place and alerting the concerned doctors when needed. This monitoring and alerting can either be manual or automated. For the automated scenario, there can be a sensor-based application acting under the influence of the emulated temperature sensors.
2. Broadcasting alert messages when magnetic field threshold in a mining area is crossed by deploying real phones with magnetometer at all the important places in the mining area. The broadcast can be either manual or automated one with the broadcast informing people working in the affected life-threatening parts of that mining area due to intense magnetic radiations, to vacate those areas immediately.
3. Remotely triggering solar battery panels of a spacecraft to open when there is ample solar light, from a virtualized system(phone) based on the emulated light sensor's readings which are the ones that are fetched from the real hardware light sensor built into the physical phone deployed on the spacecraft. There can be a cluster of spacecrafts that need to be monitored remotely in real-time to carry out this trigger either manually or in an automated fashion so that those spacecrafts don't run of power, especially during critical operations.
4. Tracking remotely the speed of given number of vehicles in real-time, meant to work in unison at the exactly same pre-defined speed for some legitimate reason. And if there is any slight difference in any of their speeds, necessary operations can be manually carried out or automated to fix the difference as per the need.
5. In a levitated train, taking necessary actions when the tolerable proximity between the railings and the levitated train's bottom is surpassed, everything remotely and in real-time. The actions can be manually taken or automated as needed.
6. Testing Android sensor-based applications on the virtualized phones. This use case is really exhaustive in itself. Same sensor-based application can be run on

many virtualized Android-x86 phones on the same physical machine at the same time in isolation. The testing can be carried out with the same sensor-based application running in several virtualized phones with different Android-x86 releases at the same time, for real-time comparisons of that application's behavior and performance on different Android-x86 releases. Same real hardware sensor readings can be used for a specific emulated sensor in many virtualized phones at the same time. All that is needed for these to happen is to pair a single real phone with many virtual phones.

7. Adding virtually any sensor defined by Android sensor subsystem to any number of virtualized phones with no restriction whatsoever as long as there is one real phone paired with these virtualized phones to feed the added sensor.
8. Experimenting the sensor-based applications on any version of Android-x86 on the virtualized phones without spending an extra penny.

The current work is accomplished for one real and one virtual phone. But, all the above use cases explained with one real and many virtual phones are very much feasible as it's just the scaling of the device server on the real phone that's needed to accept multiple socket-connection requests and provide them with the same real hardware sensor readings as opposed to the current logic of accepting one connection and sending the real hardware sensor readings to it.

One really ***out of the box*** use case of the current work would be adding an emulated sensor onto a real Android-x86 device. This is possible by adding the Sensor Emulation code for the needed sensor for the real Android-x86 device so that the device will see the sensor to be present when it's really isn't. And for this virtually added sensor on the real device, a real Android smartphone can be paired and any sensor-based application on the real Android-x86 device can be controlled using the paired real Android smartphone. The real Android-x86 device can be a tablet or a desktop with Android-x86 installed directly on its bare hardware.

Another similar ***out of the box*** use case would be with a real Android smartphone itself. The current work can also be used to emulate a sensor that's absent in a real Android smartphone and let that be driven by another paired real Android smartphone which has the emulated sensor in real.

Thus, the current work on Sensor Emulations is not limited to virtualized Android(-x86) platforms but includes real Android(-x86) platforms too. Hence, the sensor HAL based solution proposed by the current work is very universal, unique, and very useful.

Chapter 3
Requirements

The Sensor Emulation was designed and implemented with following crucial requirements under focus.

1. ***Generic*** - Sensor Emulation shouldn't be dependent on the underlying hardware at all.
2. ***Wireless*** - The work should focus on offering the real hardware sensor readings in a remote manner.
3. ***Lossless and consistent*** - Given a proper network connectivity and bandwidth, there shouldn't be any loss in the sensor readings being sent from the real to virtual device. The readings shouldn't lose their order in reaching the destination.
4. **Accurate** - The sensor readings being sent from real phone to virtual phone should be accurate in terms of the precision of the numerical values/components of the individual sensor readings.
5. ***Swift*** - The real-device-end implementation should be fast enough in sending the readings accounting for the real-time aspect of the work.
6. ***Tolerant to high-frequency transmission of sensor readings*** - The virtualized environment should be able to handle high influx of the sensor readings from the real device.
7. ***Auto-resetting and Auto-restoring*** - The whole work should be fault-tolerant in resetting and restoring the end-to-end emulation state to a sane one.
8. ***Optimized in terms of end-to-end socket-communication*** - Unnecessary send-receive socket-communication calls need to be avoided and alternatives should be found to reduce the overall network overhead.
9. ***Rigorously tested*** - The work needs to be tested and validated using all kinds of applications ranging from the simplest sensor-test applications showing just raw numbers of sensor readings to the advanced 3D sensor-intensive games which might involve the high-end graphical dependencies.
10. ***Real*** - The emulation must be as real as possible. Focus should be on having almost no differentiation at all between the real and virtualized sensors in every possible technical aspect.
11. ***Demand-less*** - No modifications to the existing emulator-compatible sensor-based application's should be demanded in order to work with the current work of Sensor Emulation.
12. ***Independent*** - Sensor Emulation must remain unaffected from all the aspects of any sensor-based application using the sensor data from the emulated sensors.

Chapter 4
Design Comparisons

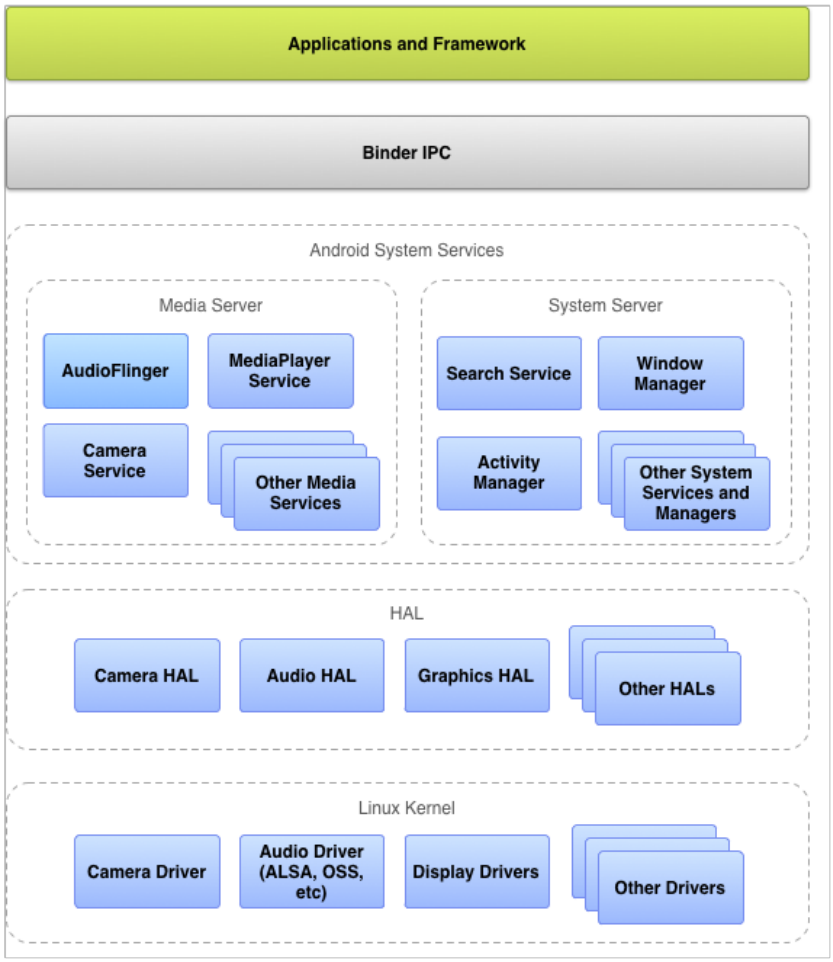


Fig. 4.1 Android Low-Level System Architecture
(Source: http://source.android.com/devices/images/system-architecture.png)

Android low-level system architecture shown in Fig. 4.1 has the below mentioned layers as listed and described in the online documentation.

Application framework

This is the level that most application developers concern themselves with. You should be aware of the APIs available to developers as many of them map 1:1 to the underlying HAL interfaces and can provide information as to how to implement your driver.

Binder IPC

The Binder Inter-Process Communication mechanism allows the application framework to cross process boundaries and call into the Android system services code. This basically allows high level framework APIs to interact with Android's system services. At the application framework level, all of this communication is hidden from the developer and things appear to "just work."

System services

Most of the functionality exposed through the application framework APIs must communicate with some sort of system service to access the underlying hardware. Services are divided into modular components with focused functionality such as the Window Manager, Search Service, or Notification Manager. System services are grouped into two buckets: system and media. The system services include things such as the Window or Notification Manager. The media services include all the services involved in playing and recording media.

Hardware abstraction layer (HAL)

The HAL serves as a standard interface that allows the Android system to call into the device driver layer while being agnostic about the lower-level implementations of your drivers and hardware. You must implement the corresponding HAL (and driver) for the particular piece of hardware that your product provides. Android does not mandate a standard interaction between your HAL implementation and your device drivers, so you have free reign to do what is best for your situation. However, you must abide by the contract defined in each hardware-specific HAL interface for the Android system to be able to correctly interact with your hardware. HAL implementations are typically built into shared library modules (`.so` files).

Linux Kernel

For the most part, developing your device drivers is the same as developing a typical Linux device driver. Android uses a specialized version of the Linux kernel with a few special additions such as wakelocks, a memory management system that is more agressive in preserving memory, the Binder IPC driver, and other features that are important for a mobile embedded platform like Android. These additions have less to do with driver development than with the system's functionality. You can use any version of the kernel that you want as long as it supports the required features, such as the binder driver. However, we recommend using the latest version of the Android kernel. For the latest Android kernel, see Building Kernels.

As a consequence, Sensor Emulation has the option to be done at different layers of Android system as shown in Fig. 4.1(previous page).

4.1 Device-Driver Level, Bottommost

To maximize the speed of Sensor Emulation, it can be done at device driver level. But, this mandates the Sensor Emulation to be driver-dependent which is in turn underlying device-dependent wherein the device being referred is the real hardware sensor. Therefore, this kills the generality of the Sensor Emulation and necessitates the work to be redone for every different device in the respective device-driver.

4.2 Application-Layer Level, Topmost

If only genericness is of interest, Sensor Emulation could have been done at application-layer but it will suffer from the obvious extra time taken for the real hardware sensor readings to reach its destination right from the device-driver level to the application-layer on the real phone and then ultimately, the virtual phone. This adds to the unwanted overall delay of transmitting the real hardware sensor readings from the real phone to virtual phone.

4.3 Levels other than at HAL

On similar lines, the work could have been done at any other layer except Hardware Abstraction Layer(HAL) but sacrificing either the generality of the Sensor Emulation or compromising the end-to-end speed of execution of the Sensor Emulation.

4.4 HAL Level

The sensor emulation is designed at system level making it a system-level Sensor Emulation. This system-level Sensor Emulation is being done at Hardware Abstraction Layer(HAL) illustrated in above **Fig. 4.1**(in one of the previous pages) and hence it's generic and portable as opposed device-driver level Sensor Emulation, given that the device driver is device-dependent as well as its vendor with respect to its specifications, etc. The HAL based implementation logic holds good regardless of any newer releases of Android that can happen in the coming years and also it's regardless of the changes that happen below the HAL like in the device-driver level, etc. This is because, the HAL is the abstract representation of the underlying hardware as opposed to the detailed and tight representation. Hence, the abstraction shall and will never break, and so is the Sensor Emulation work done at sensors HAL.

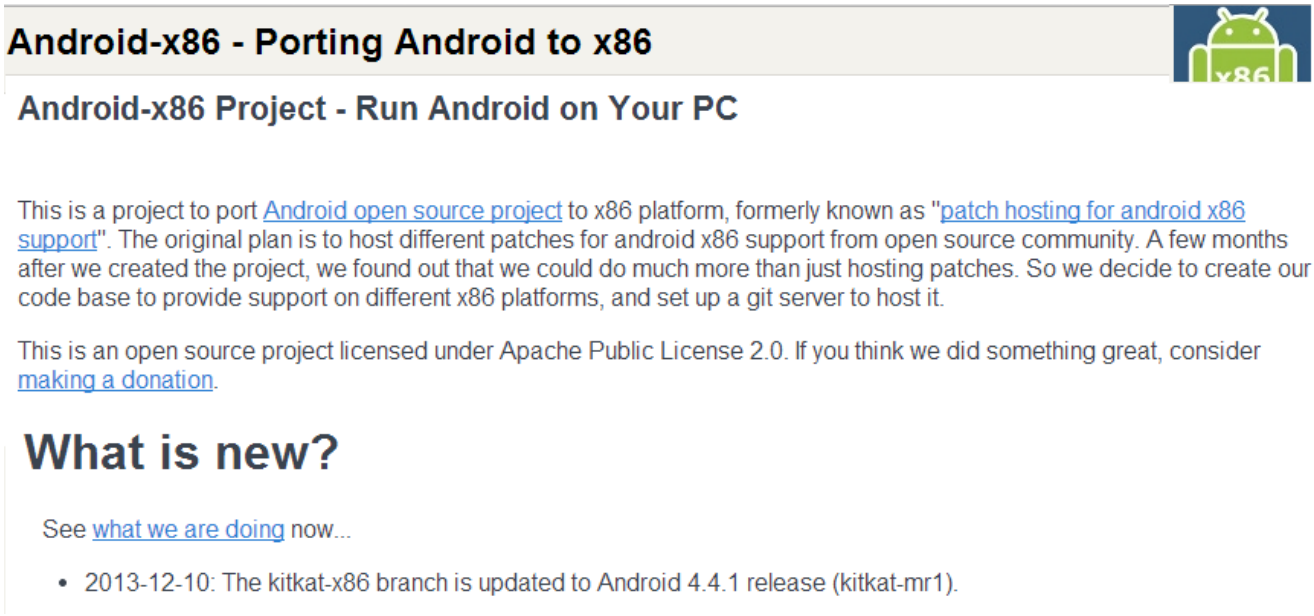
The system-level Sensor "Emulation" is way better than any application-level Sensor "Simulation", as the latter will always be running as a stand-alone application and the other sensor-based applications need to be modified to interact with this stand-alone application to get the "fake" sensor readings being generated artificially based on some logic such as input devices' data, etc. In addition, if the real device sensor readings are needed, then again this stand-alone application needs to be implemented such that it gets from the underlying abstraction layers(JNI, HAL etc) which affects the time taken in getting the real device sensor readings. And then the same stand-alone application needs to be modified for the emulated Android-x86, if the real device sensor readings are needed. Even then, the sensor-based applications running in the emulated Android-x86 need to be modified to interact with the stand-alone application running in the same emulated Android-x86 to get the real device sensor readings. Quite obviously, HAL based approach is closer to the metal than application-layer based approach of Sensor Emulation and hence the time taken in getting the real hardware sensor readings

on the real phone with application-layer based approach will more than that of with HAL based approach. Thus, this extra time taken even if in terms of nanoseconds adds to the overall time delay for the real hardware sensor readings to reach from the real phone to virtual phone. With the HAL based Sensor Emulation, there is no real burden on the application developers. The exact applications they develop for the real Android device equipped with real device sensors run with absolutely no problem, as far as the respective sensor's behavior is concerned. Thus, the system-level Sensor Emulation at HAL is proven to be efficient, portable, and universal to all the Android systems and not only just the x86 based ones.

Chapter 5
Android-x86, sensors, and HAL

5.1 Android-x86

As seen at http://www.android-x86.org/.



Android-x86 - Porting Android to x86

Android-x86 Project - Run Android on Your PC

This is a project to port Android open source project to x86 platform, formerly known as "patch hosting for android x86 support". The original plan is to host different patches for android x86 support from open source community. A few months after we created the project, we found out that we could do much more than just hosting patches. So we decide to create our code base to provide support on different x86 platforms, and set up a git server to host it.

This is an open source project licensed under Apache Public License 2.0. If you think we did something great, consider making a donation.

What is new?

See what we are doing now...

- 2013-12-10: The kitkat-x86 branch is updated to Android 4.4.1 release (kitkat-mr1).

5.2 sensors

Introduction to Android sensors as in online documentation.

Most Android-powered devices have built-in sensors that measure motion, orientation, and various environmental conditions. These sensors are capable of providing raw data with high precision and accuracy, and are useful if you want to monitor three-dimensional device movement or positioning, or you want to monitor changes in the ambient environment near a device. For example, a game might track readings from a device's gravity sensor to infer complex user gestures and motions, such as tilt, shake, rotation, or swing. Likewise, a weather application might use a device's temperature sensor and humidity sensor to calculate and report the dewpoint, or a travel application might use the geomagnetic field sensor and accelerometer to report a compass bearing.

The Android platform supports three broad categories of sensors:

- Motion sensors

  These sensors measure acceleration forces and rotational forces along three axes. This category includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors.
- Environmental sensors

  These sensors measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity. This category includes barometers, photometers, and thermometers.
- Position sensors

  These sensors measure the physical position of a device. This category includes orientation sensors and magnetometers.

You can access sensors available on the device and acquire raw sensor data by using the Android sensor framework. The sensor framework provides several classes and interfaces that help you perform a wide variety of sensor-related tasks. For example, you can use the sensor framework to do the following:

- Determine which sensors are available on a device.
- Determine an individual sensor's capabilities, such as its maximum range, manufacturer, power requirements, and resolution.
- Acquire raw sensor data and define the minimum rate at which you acquire sensor data.
- Register and unregister sensor event listeners that monitor sensor changes.

This topic provides an overview of the sensors that are available on the Android platform. It also provides an introduction to the sensor framework.

Introduction to Sensors

The Android sensor framework lets you access many types of sensors. Some of these sensors are hardware-based and some are software-based. Hardware-based sensors are physical components built into a handset or tablet device. They derive their data by directly measuring specific environmental properties, such as acceleration, geomagnetic field strength, or angular change. Software-based sensors are not physical devices, although they mimic hardware-based sensors. Software-based sensors derive their data from one or more of the hardware-based sensors and are sometimes called virtual sensors or synthetic sensors. The linear acceleration sensor and the gravity sensor are examples of software-based sensors. Table 1 summarizes the sensors that are supported by the Android platform.

Few Android-powered devices have every type of sensor. For example, most handset devices and tablets have an accelerometer and a magnetometer, but fewer devices have barometers or thermometers. Also, a device can have more than one sensor of a given type. For example, a device can have two gravity sensors, each one having a different range.

Table 1. Sensor types supported by the Android platform.

Sensor	Type	Description	Common Uses
`TYPE_ACCELEROMETER`	Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.	Motion detection (shake, tilt, etc.).
`TYPE_AMBIENT_TEMPERATURE`	Hardware	Measures the ambient room temperature in degrees Celsius (°C). See note below.	Monitoring air temperatures.
`TYPE_GRAVITY`	Software or Hardware	Measures the force of gravity in m/s^2 that is applied to a device on all three physical axes (x, y, z).	Motion detection (shake, tilt, etc.).
`TYPE_GYROSCOPE`	Hardware	Measures a device's rate of rotation in rad/s around each of the three physical axes (x, y, and z).	Rotation detection (spin, turn, etc.).
`TYPE_LIGHT`	Hardware	Measures the ambient light level (illumination) in lx.	Controlling screen brightness.

TYPE_LINEAR_ACCELERATION	Software or Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity.	Monitoring acceleration along a single axis.
TYPE_MAGNETIC_FIELD	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in μT.	Creating a compass.
TYPE_ORIENTATION	Software	Measures degrees of rotation that a device makes around all three physical axes (x, y, z). As of API level 3 you can obtain the inclination matrix and rotation matrix for a device by using the gravity sensor and the geomagnetic field sensor in conjunction with the getRotationMatrix() method.	Determining device position.
TYPE_PRESSURE	Hardware	Measures the ambient air pressure in hPa or mbar.	Monitoring air pressure changes.
TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
TYPE_RELATIVE_HUMIDITY	Hardware	Measures the relative ambient humidity in percent (%).	Monitoring dewpoint, absolute, and relative humidity.
TYPE_ROTATION_VECTOR	Software or Hardware	Measures the orientation of a device by providing the three elements of the device's rotation vector.	Motion detection and rotation detection.
TYPE_TEMPERATURE	Hardware	Measures the temperature of the device in degrees Celsius (°C). This sensor implementation varies across devices and this sensor was replaced with the TYPE_AMBIENT_TEMPERATURE sensor in API Level 14	Monitoring temperatures.

Sensor Framework

You can access these sensors and acquire raw sensor data by using the Android sensor framework. The sensor framework is part of the `android.hardware` package and includes the following classes and interfaces:

`SensorManager`
: You can use this class to create an instance of the sensor service. This class provides various methods for accessing and listing sensors, registering and unregistering sensor event listeners, and acquiring orientation information. This class also provides several sensor constants that are used to report sensor accuracy, set data acquisition rates, and calibrate sensors.

`Sensor`
: You can use this class to create an instance of a specific sensor. This class provides various methods that let you determine a sensor's capabilities.

`SensorEvent`
: The system uses this class to create a sensor event object, which provides information about a sensor event. A sensor event object includes the following information: the raw sensor data, the type of sensor that generated the event, the accuracy of the data, and the timestamp for the event.

`SensorEventListener`
: You can use this interface to create two callback methods that receive notifications (sensor events) when sensor values change or when sensor accuracy changes.

In a typical application you use these sensor-related APIs to perform two basic tasks:

- **Identifying sensors and sensor capabilities**

  Identifying sensors and sensor capabilities at runtime is useful if your application has features that rely on specific sensor types or capabilities. For example, you may want to identify all of the sensors that are present on a device and disable any application features that rely on sensors that are not present. Likewise, you may want to identify all of the sensors of a given type so you can choose the sensor implementation that has the optimum performance for your application.

- **Monitor sensor events**

  Monitoring sensor events is how you acquire raw sensor data. A sensor event occurs every time a sensor detects a change in the parameters it is measuring. A sensor event provides you with four pieces of information: the name of the sensor that triggered the event, the timestamp for the event, the accuracy of the event, and the raw sensor data that triggered the event.

Sensor Availability

While sensor availability varies from device to device, it can also vary between Android versions. This is because the Android sensors have been introduced over the course of several platform releases. For example, many sensors were introduced in Android 1.5 (API Level 3), but some were not implemented and were not available for use until Android 2.3 (API Level 9). Likewise, several sensors were introduced in Android 2.3 (API Level 9) and Android 4.0 (API Level 14). Two sensors have been deprecated and replaced by newer, better sensors.

Table 2 summarizes the availability of each sensor on a platform-by-platform basis. Only four platforms are listed because those are the platforms that involved sensor changes. Sensors that are listed as deprecated are still available on subsequent platforms (provided the sensor is present on a device), which is in line with Android's forward compatibility policy.

Table 2. Sensor availability by platform.

Sensor	Android 4.0 (API Level 14)	Android 2.3 (API Level 9)	Android 2.2 (API Level 8)	Android 1.5 (API Level 3)
TYPE_ACCELEROMETER	Yes	Yes	Yes	Yes
TYPE_AMBIENT_TEMPERATURE	Yes	n/a	n/a	n/a
TYPE_GRAVITY	Yes	Yes	n/a	n/a
TYPE_GYROSCOPE	Yes	Yes	n/a[1]	n/a[1]
TYPE_LIGHT	Yes	Yes	Yes	Yes
TYPE_LINEAR_ACCELERATION	Yes	Yes	n/a	n/a
TYPE_MAGNETIC_FIELD	Yes	Yes	Yes	Yes
TYPE_ORIENTATION	Yes[2]	Yes[2]	Yes[2]	Yes
TYPE_PRESSURE	Yes	Yes	n/a[1]	n/a[1]
TYPE_PROXIMITY	Yes	Yes	Yes	Yes
TYPE_RELATIVE_HUMIDITY	Yes	n/a	n/a	n/a
TYPE_ROTATION_VECTOR	Yes	Yes	n/a	n/a
TYPE_TEMPERATURE	Yes[2]	Yes	Yes	Yes

[1] This sensor type was added in Android 1.5 (API Level 3), but it was not available for use until Android 2.3 (API Level 9).

[2] This sensor is available, but it has been deprecated.

5.3 Hardware Abstraction Layer(HAL)

As it's mentioned in the online documentation here is the description about HAL, again for reference.

Hardware abstraction layer (HAL)

The HAL serves as a standard interface that allows the Android system to call into the device driver layer while being agnostic about the lower-level implementations of your drivers and hardware. You must implement the corresponding HAL (and driver) for the particular piece of hardware that your product provides. Android does not mandate a standard interaction between your HAL implementation and your device drivers, so you have free reign to do what is best for your situation. However, you must abide by the contract defined in each hardware-specific HAL interface for the Android system to be able to correctly interact with your hardware. HAL implementations are typically built into shared library modules (`.so` files).

5.4 Android Sensors HAL

Here's a brief description of Android Sensors for reference from the online documentation.

Sensor axis definition
Accuracy
Power consumption
HAL release cycle
Interaction
Concurrent running
Interaction with suspend mode
Sensor fusion and virtual sensors
HAL interface

Android sensors give applications access to a mobile device's underlying base sensor(s): accelerometer, gyroscope, and magnetometer. Manufacturers develop the drivers that define additional composite sensor types from those base sensors. For instance, Android offers both calibrated and uncalibrated gyroscopes, a geomagnetic rotation vector and a game rotation vector. This variety gives developers some flexibility in tuning applications for battery life optimization and accuracy.

The Sensors Hardware Abstraction Layer (HAL) API is the interface between the hardware drivers and the Android framework; the Sensors Software Development Kit (SDK) API is the interface between the Android framework and the Java applications. Please note, the Sensors HAL API described in this documentation is not identical to the Sensors SDK API described on developer.android.com. For example, some sensors that are deprecated in the SDK may still exist in the HAL, and vice versa.

Similarly, audio recorders, Global Positioning System (GPS) devices, and accessory (pluggable) sensors are not supported by the Android Sensors HAL API described here. This API covers sensors that are physically part of the device only. Please see the Audio, Location Strategies, and the Accessories section for information on those devices.

Sensor fusion and virtual sensors

Many composite sensor types are or can be implemented as virtual sensors from underlying base sensors on the device. Examples of composite sensors types include the rotation vector sensor, orientation sensor, step detector and step counter.

From the point of view of this API, these virtual sensors **must** appear as real, individual sensors. It is the responsibility of the driver and HAL to make sure this is the case.

In particular, all sensors must be able to function concurrently. For example, if defining both an accelerometer and a step counter, then both must be able to work concurrently.

5.5 HAL Data structures

Every piece of hardware for an Android device needs to have a standard Android hardware module interface to be defined and implemented with the default methods(functions) for controlling it.

5.5.1 Hardware module representation

As in `hardware/hardware.h.(<android_src_path>/hardware/libhardware/include/hardware/hardware.h)`

```
/**
 * Every hardware module must have a data structure named HAL_MODULE_INFO_SYM
 * and the fields of this data structure must begin with hw_module_t
 * followed by module specific information.
 */
typedef struct hw_module_t {
    /** tag must be initialized to HARDWARE_MODULE_TAG */
    uint32_t tag;

    /**
     * The API version of the implemented module. The module owner is
     * responsible for updating the version when a module interface has
     * changed.
     *
     * The derived modules such as gralloc and audio own and manage this field.
     * The module user must interpret the version field to decide whether or
     * not to inter-operate with the supplied module implementation.
     * For example, SurfaceFlinger is responsible for making sure that
     * it knows how to manage different versions of the gralloc-module API,
     * and AudioFlinger must know how to do the same for audio-module API.
     *
     * The module API version should include a major and a minor component.
     * For example, version 1.0 could be represented as 0x0100. This format
     * implies that versions 0x0100-0x01ff are all API-compatible.
     *
     * In the future, libhardware will expose a hw_get_module_version()
     * (or equivalent) function that will take minimum/maximum supported
     * versions as arguments and would be able to reject modules with
     * versions outside of the supplied range.
     */
    uint16_t module_api_version;
#define version_major module_api_version
    /**
     * version_major/version_minor defines are supplied here for temporary
     * source code compatibility. They will be removed in the next version.
     * ALL clients must convert to the new version format.
     */

    /**
     * The API version of the HAL module interface. This is meant to
     * version the hw_module_t, hw_module_methods_t, and hw_device_t
     * structures and definitions.
     *
     * The HAL interface owns this field. Module users/implementations
     * must NOT rely on this value for version information.
     *
     * Presently, 0 is the only valid value.
     */
    uint16_t hal_api_version;
#define version_minor hal_api_version

    /** Identifier of module */
    const char *id;

    /** Name of this module */
    const char *name;
```

```
    /** Author/owner/implementor of the module */
    const char *author;

    /** Modules methods */
    struct hw_module_methods_t* methods;

    /** module's dso */
    void* dso;

    /** padding to 128 bytes, reserved for future use */
    uint32_t reserved[32-7];

} hw_module_t;
```

5.5.2 Hardware module methods representation

As in **hardware/hardware.h**.

```
typedef struct hw_module_methods_t {
    /** Open a specific device */
    int (*open)(const struct hw_module_t* module, const char* id,
            struct hw_device_t** device);

} hw_module_methods_t;
```

5.5.3 Hardware device representation

As in **hardware/hardware.h**.

```
/**
 * Every device data structure must begin with hw_device_t
 * followed by module specific public methods and attributes.
 */
typedef struct hw_device_t {
    /** tag must be initialized to HARDWARE_DEVICE_TAG */
    uint32_t tag;

    /**
     * Version of the module-specific device API. This value is used by
     * the derived-module user to manage different device implementations.
     *
     * The module user is responsible for checking the module_api_version
     * and device version fields to ensure that the user is capable of
     * communicating with the specific module implementation.
     *
     * One module can support multiple devices with different versions. This
     * can be useful when a device interface changes in an incompatible way
     * but it is still necessary to support older implementations at the same
     * time. One such example is the Camera 2.0 API.
     *
     * This field is interpreted by the module user and is ignored by the
     * HAL interface itself.
     */
    uint32_t version;

    /** reference to the module this device belongs to */
    struct hw_module_t* module;
```

```
    /** padding reserved for future use */
    uint32_t reserved[12];

    /** Close this device */
    int (*close)(struct hw_device_t* device);

} hw_device_t;
```

5.6 Sensor HAL Representation

5.6.1 Sensor Representation

As in `hardware/sensors.h`.

```
struct sensor_t {

    /* Name of this sensor.
     * All sensors of the same "type" must have a different "name".
     */
    const char*     name;

    /* vendor of the hardware part */
    const char*     vendor;

    /* version of the hardware part + driver. The value of this field
     * must increase when the driver is updated in a way that changes the
     * output of this sensor. This is important for fused sensors when the
     * fusion algorithm is updated.
     */
    int             version;

    /* handle that identifies this sensors. This handle is used to reference
     * this sensor throughout the HAL API.
     */
    int             handle;

    /* this sensor's type. */
    int             type;

    /* maximum range of this sensor's value in SI units */
    float           maxRange;

    /* smallest difference between two values reported by this sensor */
    float           resolution;

    /* rough estimate of this sensor's power consumption in mA */
    float           power;

    /* this value depends on the trigger mode:
     *
     *   continuous: minimum sample period allowed in microseconds
     *   on-change : 0
     *   one-shot  :-1
```

```
     *   special   : 0, unless otherwise noted
     */
    int32_t         minDelay;

    /* reserved fields, must be zero */
    void*           reserved[8];
};
```

5.6.2 Sensor Module Representation

As in **hardware/sensors.h**.

```
/**
 * Every hardware module must have a data structure named HAL_MODULE_INFO_SYM
 * and the fields of this data structure must begin with hw_module_t
 * followed by module specific information.
 */
struct sensors_module_t {
    struct hw_module_t common;

    /**
     * Enumerate all available sensors. The list is returned in "list".
     * @return number of sensors in the list
     */
    int (*get_sensors_list)(struct sensors_module_t* module,
            struct sensor_t const** list);
};
```

5.6.3 Sensor Event Representation

As in **hardware/sensors.h.**

```
/**
 * Union of the various types of sensor data
 * that can be returned.
 */
typedef struct sensors_event_t {
    /* must be sizeof(struct sensors_event_t) */
    int32_t version;

    /* sensor identifier */
    int32_t sensor;

    /* sensor type */
    int32_t type;

    /* reserved */
    int32_t reserved0;

    /* time is in nanosecond */
    int64_t timestamp;
```

```
    union {
        float           data[16];

        /* acceleration values are in meter per second per second (m/s^2) */
        sensors_vec_t   acceleration;

        /* magnetic vector values are in micro-Tesla (uT) */
        sensors_vec_t   magnetic;

        /* orientation values are in degrees */
        sensors_vec_t   orientation;

        /* gyroscope values are in rad/s */
        sensors_vec_t   gyro;

        /* temperature is in degrees centigrade (Celsius) */
        float           temperature;

        /* distance in centimeters */
        float           distance;
```

```
        /* light in SI lux units */
        float           light;

        /* pressure in hectopascal (hPa) */
        float           pressure;

        /* relative humidity in percent */
        float           relative_humidity;

        /* step-counter */
        uint64_t        step_counter;

        /* uncalibrated gyroscope values are in rad/s */
        uncalibrated_event_t uncalibrated_gyro;

        /* uncalibrated magnetometer values are in micro-Teslas */
        uncalibrated_event_t uncalibrated_magnetic;
    };
    uint32_t        reserved1[4];
} sensors_event_t;
```

5.6.4 Common Sensor HAL Calls

Below is a list of common sensor HAL function calls that happen as mentioned in the online documentation.

HAL interface

These are the common sensor calls expected at the HAL level:

1. *getSensorList()* - Gets the list of all sensors.
2. *activate()* - Starts or stops the specified sensor.
3. *batch()* - Sets parameters to group event data collection and optimize power use.
4. *setDelay()* - Sets the event's period in nanoseconds for a given sensor.
5. *flush()* - Flush adds an event to the end of the "batch mode" FIFO for the specified sensor and flushes the FIFO.
6. *poll()* - Returns an array of sensor data.

getSensorList(sensor_type)

Provide the list of sensors implemented by the HAL for the given sensor type.

activate(sensor, true/false)

```
int (*activate)(struct sensors_poll_device_t *dev,
        int handle, int enabled);
```

batch(sensor, batching parameters)

```
int (*batch)(struct sensors_poll_device_1* dev,
       int handle, int flags, int64_t period_ns, int64_t timeout);
```

setDelay(sensor, delay)

```
int (*setDelay)(struct sensors_poll_device_t *dev,
        int handle, int64_t period_ns);
```

flush()

```
int (*flush)(struct sensors_poll_device_1* dev, int handle);
```

poll()

```
int (*poll)(struct sensors_poll_device_t *dev,
        sensors_event_t* data, int count);
```

Chapter 6
Sensor Emulation Software Architecture

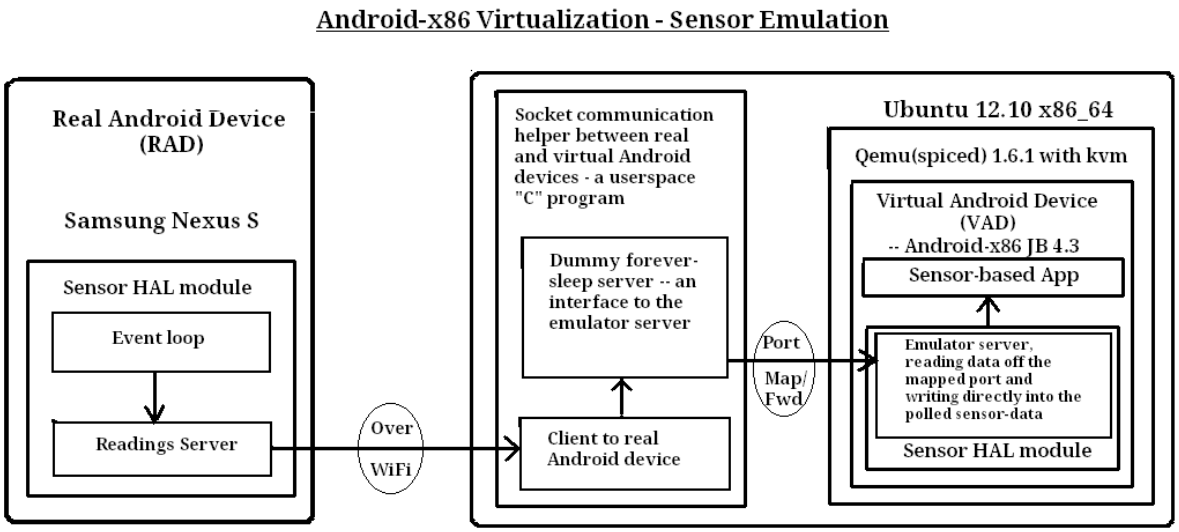


Fig. 6 Sensor Emulation Software Architecture

Note: The above architecture remains same in the case of a remote-server in place of RAD except that the remote-server generates the artificial sensor readings for each of the sensor being queried for the emulation. The method used to generate artificial readings is a simple one based on random number generation, as of now.

The above Fig. 6 depicts the Sensor Emulation Software Architecture with the three main components of the emulation work in action.

1. Sensor HAL module on the physical phone(Real Android Device - RAD).
2. Sensor HAL module on the virtual phone(Virtual Android Device - VAD).
3. Intermediate userspace program facilitating the transmission of real sensor readings from the physical phone to the virtual phone.

A complete cycle of Sensor Emulation works as under.

1. Real hardware sensor readings get processed in the event loop of Sensor HAL module of RAD.
2. The processes readings are given to the readings server part of the same Sensor HAL module through a shared buffer(pipe).
3. The readings server sends those readings to the client to real device over wireless network. The wireless transmission accounts for the ***remoteness***.

4. The client to real device which is part of the intermediate userspace program, sends the received readings to the dummy forever-sleep server which is also running as part of the same intermediate userspace program.
5. Due to port mapping(forwarding), the readings sent to the dummy forever sleep server automatically are forwarded to the emulator server for the specific sensor. The emulator server will be running as part of the Sensor HAL module of VAD.
6. The emulator server extracts the numerical sensor readings from the received readings network buffer onto a global buffer.
7. The periodically polled function of the Sensor HAL module of VAD reads the global buffer and prepares the array of sensor readings with the respective sensor types and returns this array as a set of sensor events along with the number of sensor events.
8. The sensor-based application running in VAD will ultimately receive the readings from the Sensor HAL via all the intermediate layers and thereon the application can use the sensor readings just like it does on a RAD.

The above repetitive cycle is applicable independently for each of the sensors being emulated. Both in RAD and VAD, the existing sensors specific code belonging to Sensor HAL module while initializing the respective sensor, will spawn these clients and servers as independent threads. These clients and servers will be dedicated to the respective sensors to let the unaffected clients and servers to continue independently, if one of the clients or servers get affected by any error.

End-to-end, the sensor readings are being dealt with using unique buffers. This end-to-end uniqueness refers to everything from the buffer used for the internal processing of the sensor readings of each sensor within the Sensor HAL module in RAD, the network buffer used for packing and sending the sensor readings processed within the RAD, receiving the packed and sent readings in the client to real device, parsing and extracting the received readings within the Sensor HAL module for each sensor in VAD. This uniqueness accounts greatly for the incredible speed of the overall Sensor Emulation since there will be no common network buffer that would be otherwise be dominated by the readings from the sensors with higher sampling frequencies leaving the readings of relatively slower sampling frequencies to starve from reaching their destination.

There is a symmetry in the design of the Sensor HAL module workings between RAD and VAD for each of the sensors being emulated with only one difference - RAD hosts the servers sending the real hardware sensor readings while VAD also hosts the servers but receiving those sent real hardware sensor readings. The servers on VAD can’t be clients as they are receiving and not sending as opposed to usual server logic of sending data and not receiving data. This is because, the readings are received via the mapped ports inside the Sensor HAL module of VAD and only servers can listen to these mapped ports and not clients.

It's necessary to note that the intermediate userspace program is mentioned specifically being in "C". The real reason for choosing it to be in "C" is that for the Sensor Emulation to be real-time, speed is very crucial. And it's known that an efficient implementation of an algorithm in "C" scores over an equivalently efficient implementation of the same algorithm in any high-level language such as Java, etc - the difference in speed may be fraction of a fraction of a fraction of a nanosecond, but that ***minutest*** speed difference must not and will not be overlooked as the current work of Sensor Emulation is aimed to be **real-time**. And the reason for mentioning it in the software architecture diagram itself is to avoid any confusion over the choice of implementation language for the intermediate userspace program when the speed is of utmost importance.

Sensors being emulated in the current work is all the sensors that are supported by **Android Sensors HAL API** which are necessarily ***physically part*** of the real Android device, Samsung Nexus S smartphone.

Real sensors

1. Accelerometer
2. Magnetometer
3. Light
4. Proximity
5. Gyroscope

Virtual sensors

6. Corrected Gyroscope
7. Linear Acceleration
8. Gravity
9. Orientation
10. Rotation Vector

Remaining sensors supported by Android Sensors HAL API are:

1. Temperature
2. Ambient temperature
3. Pressure
4. Relative Humidity

Detailed information about all the Android sensors is provided in the earlier Chapter 5.

The Audio recorders(microphones, etc), Global Positioning System (GPS) devices, and accessory (pluggable) sensors are not physically part of the real Android device and hence are not supported by Android Sensors HAL API. Thus, these are not being emulated. However, for these sensors not supported by Android Sensors HAL API, the existing Sensor Emulation logic works as the logic is just socket-communication based transmission of sensor readings from RAD to VAD and has no dependency on Android Sensors HAL API as such. In fact, for these sensors unsupported by Android Sensors

HAL API, the same client-server driven socket-communication between RAD and VAD can be used for the Sensor Emulation, in the related Android HAL module but not in Android Sensors HAL module.

It should be noted that this work is **precisely** about **Sensor Emulation** and not any Hardware Emulation in general. So, emulating entire hardware components which have sensor(s) as one of their subcomponents is out of the scope of the current work. For example, emulating entire Camera hardware of Android which has an image sensor as one of its subcomponents, is just out of the question.

Chapter 7
Design Considerations

The design considered comprises of three main components.

1. The device server on the real Android device.
2. The userspace “C” program in the host.
3. The emulator server on the virtual Android device.

Socket-communication based interactions between these three components facilitate the necessary Sensor Emulation on the virtualized Android-x86. All these socket-communication are designed to be part of the Sensor HAL module on both the real and virtual Android devices, in addition to the client-server logic implemented in the userspace “C” program targeted for the host running the virtualized Android-x86 on Qemu.

Inside the sensor HAL module, the reads and writes of the sensor readings are designed to be efficient, fast, reliable and real-time. In the device sensor HAL module, the each of the existing sensor event loop is modified to notify the respective sensor readings to the corresponding real device server running as part of the same device sensor HAL module via one of the inter-thread communication mechanisms, pipes. The real device server is designed to be as efficient as possible and robust at the same time. The efficiency is in terms of the absence of synchronous polling for change in the device sensor readings, but asynchronous as pipes are being used for sharing sequentialized sensor data. Robustness is in terms of being fault-tolerant to the failures in socket-communication errors. Should there be any socket communication errors, the respective socket communication logic is designed so as to reset the socket-server to a sane state saved previously, which is usually the entry point of the corresponding sensor server thread

Each sensor has its own dedicated pair of client-server end-to-end. This design choice ensures that each healthy sensor client/server remains unaffected by any other error-prone sensor client(s)/server(s). This also implies that the device sensor server for a sensor runs on demand only when that sensor is enabled and active. In addition, the dedicated sensor servers rule out the very need to parse every incoming network buffer/packet to detect which sensor’s reading it is which would be the case when a single sensor server is sending the aggregated sensors readings compromising on the number of sensor device server threads in the device sensor HAL module with an intent of reducing the number of socket TCP/IP connections. This dedication also simplifies the entire design to a greater extent as it doesn’t necessarily mandate the event loop of each

of the sensor to be filling the same shared buffer in a synchronized manner with the respective sensor readings, given that each sensor is implemented as a separate hardware module. The sheer amount of synchronization overhead involved in such a scenario prevents the sensor readings to be sent and eventually available on the emulated Android-x86, in real-time at high-speed. Hence, the design choice of the dedicated client-server for each of the sensors end-to-end is proven to be efficient and robust as claimed.

As mentioned in the software architecture chapter, userspace program needs to be implemented in "C" as opposed to in any high-level language such as Java or even scripting languages such as Python, etc, for not compromising on even the minutest of the minutest time delay for the real hardware sensor readings to reach from the RAD to VAD.

The userspace "C" program is designed to be having a dummy server which acts as a gateway/interface/backdoor to the emulator server. There are dedicated dummy servers for each of the emulator sensor servers. These dummy servers are made to accept connection with respect to a mapped port from the host to the guest(Android-x86) and then put to forever-sleep by some suitable mechanism like waiting on a semaphore which never gets signaled by other thing. So, when the client-to-device or client-to-remote-server wants to send the received sensor readings to the emulator server, it sends those readings to the respective dummy server for the corresponding sensor at a particular port. Due to the port-mapping/forwarding mechanism, the same reading gets forwarded from the host to guest and thus it reaches the respective emulator sensor server listening to the mapped port. This design considered over explicit send-receive between the userspace "C" program and the emulator server helps in reducing the overhead of making explicit send/recv socket-communication based calls in the userspace "C" program. And also, this rules out any need for the userspace "C" program and the emulator servers to know each others' machine's ip-address and also the port to be used for the socket-based communication. Thus making the interaction between the userspace "C" program and the emulator server generic.

The emulator server mentioned above is designed as a server though it receives but doesn't send anything as opposed to the regular notion of server sending and client receiving it. The reasons for that to be a server is that the sensor readings are got through the mapped port from the host(Ubuntu) of the emulator to the guest(emulated Android-x86) and hence it needs to listen to the mapped port and thus it needs to be a server receiving readings and not a client.

To account for the high-frequency real Android sensors, Gyroscope and Accelerometer, in that order, as opposed to the relatively low-frequency ones which are the remaining three real Android sensors(magnetic, light and proximity), their emulator server code are designed to be fine-tuned as per the rate at which the gyroscope and accelerometer

sensor readings arrive from the real device to the emulator. This fine tuning is a ***must*** to account for the ***accurate, lossless***, and ***real*** aspects of Sensor Emulation, specifically for accelerometer and gyroscope sensors. Otherwise, if fine-tuning is not needed, one needs to compromise on those three important aspects of the current work. The sampling frequencies of accelerometer and gyroscope on the real phone can be brought down to match that of the other slower sensors so that send-one-receive-one will be the common logic for all of the client-server pairs of all the sensors end-to-end - compromises the losslessness and accurateness of Sensor Emulation, but some of the sensor readings will be lost within the device itself due to the reduced sampling frequency though the underlying real hardware sensor will still be sensing at its original rate. Instead of modifying the real hardware sensor's sampling frequency, the sampling frequency of the emulated sensor can be brought down but this is useless - compromises realness, accurateness, and losslessness since the emulated sensor is no longer working at the same sampling frequency as that of the respective real hardware sensor, hence loss of some of the readings, and thus not accurate as well when compared to the behavior of the real hardware sensor. In short, ***fine-tuning*** is a **must** for the sensors(accelerometer and gyroscope) with relatively high sampling frequencies since the current work of Sensor Emulation **must** be ***accurate***, ***lossless*** and ***real,*** in addition to other requirements.

The fine-tuning is in terms of the amount of the readings received at a time and the sleep delay for these emulator servers. However, still, the gyroscope and accelerometer real device servers and the userspace "C" programs are designed to send the respective sensor readings one at a time, as they are not the points of bottlenecks but the corresponding emulator sensor servers. The fine-tuning is necessary since the gyroscope and accelerometer sensor readings get queued up in the network buffer even when they are being sent one at time by their respective real device sensor servers when their emulator servers are slow in receiving the readings one at a time. The slowness could be due to the inherent overhead of the network-based communication between the host and the emulator, the relatively slow the sensor server loop due to the extra checks on the network-state, packet received and so on. This slowness would make the Sensor Emulation for gyroscope and accelerometer slow and hence affects the sensor-initiated responsiveness of the gyroscope and/or accelerometer sensor-based Android applications running on the emulator.

As a consequence, the gyroscope emulator server is designed to be receiving a bunch of pre-defined number of readings and so is the accelerometer emulator server. But, there is a difference in the number of readings received by these servers and also the interval at which these readings are received. The interval is short for gyroscope when compared to accelerometer, since gyroscope is too sensitive and faster than accelerometer on the real Android device. Apart from this design customization for these two sensors, the way in which the bunch of readings received is processed is also customized and tuned for only these two sensors as opposed to other three real sensors.

Since, the periodically polled sensor event data loop function is designed to process only one reading at a time for speed and responsiveness, there is an intermediate buffer in terms of pipe(inter-thread communication mechanism) dedicated to these two sensors separately. One for the accelerometer and another for gyroscope. As a result, these bunch of readings are written by the respective sensor servers onto the specific sensor pipes(accelerometer-pipe and gyroscope-pipe) in a one go and the periodically polled sensor event loop function reads off one reading at a time from the respective pipe and returns it as the corresponding sensor's data to the needed sensor subsystem. This receive-many-process-one design consideration has a huge speed benefit over receive-one-process-one which became apparent during the evaluation.

The entire sensor emulation code is designed to be comprehensively and thoroughly logged. So, any developer will find the code to be friendly while debugging, if and when the need be. These logs are macro enabled and hence can be enabled or disabled as per the need. Each sensor has separate logs to ease the debugging. All the logs are designed to be file-based and to be in separate files to let the developers follow each sensor behavior easily. The standard built-in Android logging technique is not used since the logcat output is not persistent across the booting of the system unless redirected to a file or something similar, especially when there is a crash that brings down the entire Android system triggering a reboot. So, again this ends up indirectly in a file-based logging mechanism which is what the current file-based logging does in the first place.

The design decision to do the Sensor Emulation work at HAL level provides an edge due to the fact that it's closer to the hardware on the real Android device as opposed to the same scenario with application-level Sensor Emulation. The direct and most important consequence of this is the entire Sensor Emulation at HAL needing to be done in C/C++ which again has an obvious edge over any application-level Sensor Emulation in terms of performance, which will be written in Java and hence the overhead of reaching down all the way from the application layer to the HAL through other intermediate layers to get the real device readings. This overhead can be negligible in general, but not when even nanoseconds of difference is important. This definitely is important when claiming the Sensor Emulation to be real-time, which is indeed one of the main highlights of the current system-level Sensor Emulation.

Chapter 8
Demo

8.1 Paired Real Device scenario

Paired-real-device-scenario - Sensor Emulation using the real device sensor readings received over the network(WiFi).

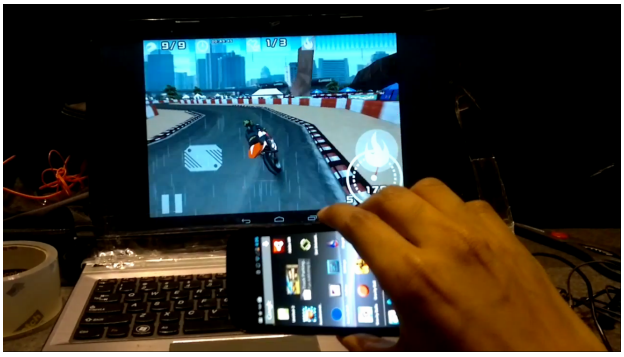


Fig. 8.1.1 Snapshot 1 - Emulated Accelerometer in action

Fig. 8.1.2 Snapshot 2 - Emulated Gyroscope in action

8.2 Remote server scenario

Remote-server-scenario - Sensor Emulation using artificially generated sensor readings fetched from a remote server, supposedly running anywhere as long as it's ip-address is known and public.

Fig. 8.2.1 Snapshot 3 - All 10 sensors in action - remote server scenario

Chapter 9
Implementation

The HAL module-specific implementation is split based on the real Android sensors and the virtual Android sensors resulting from the Android Sensor Fusion technique. All of the sensors that are present in the real Android device, Samsung Nexus S are being emulated. They are as under.

Real Sensors - 1) Accelerometer 2) Magnetometer 3) Light 4) Proximity 5) Gyroscope
Virtual Sensors(constructed based on real ones using Android Sensor Fusion method) - 6) Linear Acceleration 7) Orientation 8) Gravity 9) Rotation Vector 10) Corrected Gyroscope.

For both the device and the emulated Android-x86, the source for the real sensors are present under **<android_src_path>/hardware/libsensors.** The source for the virtual sensors are under **<android_src_path>/frameworks/service/sensorservice.** The exact source paths are as under for both the device and the emulated Android-x86: **Real sensors(all under <android_src_path>/hardware/libsensors/)- AkmSensor.cpp, GyroSensor.cpp, InputEventReader.cpp, LightSensor.cpp, ProximitySensor.cpp. Virtual sensors(all under <android_src_path>/frameworks/service/sensorservice/) - LinearAccelerationSensor.cpp, RotationVectorSensor.cpp, SensorFusion.cpp, SensorService.cpp, CorrectedGyroSensor.cpp, GravitySensor.cpp, OrientationSensor.cpp**

The real sensors' code is deployed on RAD as a single library, **sensors.herring.so** under **/system/lib/hw/** while as **sensors.emu.so** on VAD under **/system/lib/hw/**. The virtual sensors' code is deployed on RAD and VAD as a single library, **libsensorservice.so**. under **/system/lib/**. These libraries are built as according to the target platforms from more or less similar sources since only one of them will be acting as a server and the other acting as a server which receives readings instead of sending any, from the mapped port. The only obvious difference between the sources for the device and the emulated Android-x86 is that the device side of the Android sources implement servers providing the device sensor readings and the emulated Android-x86 side of Android sources also implement a server but receives the device sensor readings via the mapped port. The emulated Android-x86 side must be a server against the usual notion of server only sending data and not receiving any, because it has to **listen** to a mapped port. The userspace "C" program has two sources:

SensorEmulationClientServer.c, SensorEmulationRemoteServer.c. From here on, unless and otherwise mentioned, the phrase "The userspace "C" program" refers to **SensorEmulationClientServer.c** specific to the client connecting to a real Android device. **SensorEmulationRemoteServer.c** will be referred at the end. For connecting to a RAD, **SensorEmulationClientServer.c** needs to be compiled with **DEVICE_READINGS** defined as a pre-processor macro and for connecting to a remote server implemented by **SensorEmulationRemoteServer.c**, **SensorEmulationClientServer.c** needs to be compiled with **REMOTE_SERVER_READINGS** as a pre-processor macro.

The userspace "C" program intended to be as an intermediary between the device and the emulated Android-x86, is implemented with as many clients as the real device sensors, which connect to the respective device readings server dedicated to a particular sensor(real or virtual). These clients to real device will have their server counterparts which are the dummy forever-sleep servers whose duty is to accept any one incoming connection(from its client counterpart) and go for an infinite sleep, specifically, waiting on a semaphore which never gets signalled. The dummy forever-sleep server concept is crucial in reducing the overall time consumption. The reduction is due to the lack of the need for any explicit receive-send for each of the device readings got, from the host to the guest. All that is being done is that the connected client writes to a connected socket of the dummy forever-sleep server which is in fact listening to a port that is mapped onto the same numbered port inside the emulated Android-x86. The device's ip-address is the only thing need to be known in advance for the user-space "C" program for the sake of its client-to-device. The ip-address will be read from a pre-determined file expected to be in the current directory of the userspace "C" program(process). The syntax is just plain one-liner having the ip-address in its usual layout which is xxx.xxx.xxx.xxx terminated by a new-line. The port-mapping facility employed also rules out any know-in-advance ip-addresses between the host and guest - the host and the guest neither need to be aware of each others' ip-address nor the ports being listened to. The servers on the host listen to the host ports and the servers on the guest to the guest ports - purely based on the **localhost.**

As of now, the entire concept of Sensor Emulation assumes that each sensor's servers and clients use only pre-determined ports assigned to them uniquely. This ensures that there is no intermix of the sensor readings and hence the sensor data transmission is streamlined end-to-end. The assigned ports are as under. **Accelerometer:5000, Magnetometer:5001, Light:5002, Proximity:5003, Gyroscope:5004, Orientation:5005, Corrected Gyroscope:5006, Gravity:5007, Linear Acceleration:5008, Rotation Vector:5009.** However, if one-server-many-clients needs to be implemented, then the hard-coded port numbers need to changed just by offsetting by 10(as there are 10 sensors being emulated) and letting know all the dependent client/servers/dummy-servers the exact port to be used for the socket-communication. Thus, the above idea is still scaleable to any number of clients.

The delay chosen in each of the servers, clients, event loop everywhere is such that the underlying system's resource utilization is nominal and the sensor readings retain their real-time nature, end-to-end, RAD-to-VAD. All the servers and clients are implemented as **pthread**s of the respective process on the particular system(RAD, VAD, or Ubuntu). As far as the readings are concerned, they are sent from the device to the emulated Android-x86 as a buffer of a pre-determined maximum length and is a "C" string with a specific pattern for each of the sensors. For example, accelerometer will have a x-y-z triplet in its readings. So, it's reading will be transmitted in a pattern that looks like "0.000000|0.000000|0.000000", wherein the pipe(|) symbol serves as the delimiter for the individual component of the triplet. So, there is no fixed length for each of the component but the entire pattern has a maximum limit which is the pre-determined maximum length. For the maximum genericness, the userspace "C" program doesn't do anything with the sensors readings buffer of a particular pattern received from the device. Instead, it just sends it to the emulator via the dummy server based on the mapped port. The emulator server for the specific sensors will be responsible to interpret the readings correctly by parsing the buffer of a particular pattern. This very don't-touch-just-send nature of the userspace "C" program makes it very generic and portable - for any new sensor being emulated, all that needs to be done is to increase the number of sensors defined in it and update the sensors' names list. Rest of the work will be done on the device-side code as well as the emulator-side code.

For majority of the sensors, i.e., 8 out 10 sensors being emulated, the send-one-receive-one for the sensor readings are applicable. The only two sensors which are having servers that send one at a time on the real device and receive a bunch in the emulator are the gyroscope and accelerometer. The send-one-receive-one greatly simplifies the overall design for the rest of the 8 sensors as there is no separate processing criteria for each of the single sensor readings received apart from the parsing the pipe-delimited sensor readings pattern. However, gyroscope and accelerometer demanded send-one-receive-many as these two sensors have higher sampling frequencies than the other eight sensors. Hence, the device sensor servers for these two sensors send the respective sensor readings at a higher rate than the emulator servers can process. As a consequence, **sensors_emu.c** implements the device sensor servers for these two sensors with receive many logic so that the sensor readings sent at a high rate even though one at a time, don't spend much time in the network buffer resulting in slow responsive emulated sensors. The implementation involves writing the bunch of readings received onto a shared pipe by the respective emulator server and the periodical polling function(**dummy_poll()**) reads the sensor readings off the shared pipe, one reading at a time to keep the polling function simple and quick. Each of the two sensors, gyroscope and accelerometer have their own shared pipes(**gyro_pipe** and **accel_pipe**, respectively) for these inter-thread communication of the sensor readings. To make sure that these don't affect the real-time nature of the emulation(by blocking the servers and/or the polling function when pipe is full for server to write or empty for the polling function to read), both **gyro_pipe** and **accel_pipe** are made

non-blocking(O_NONBLOCK) using explicit **`fcntl()`** calls after piping them with **`pipe()`** as **`pipe2()`** isn't available in Android.

The usage of TCP over UDP for the exchange of readings end-to-end does ensure consistency and absolutely no loss of the readings as far as the network connectivity staying in tact. The only scenario wherein there can be loss of readings is that within the device itself, the pipe buffer which is written into by the respective device sensor event loop might get full causing some of the readings not being transmitted at all from the device itself over the network towards any connected client - an unfortunate scenario beyond one's control. If there are any loss of sensor readings within the device itself which prevents from transmitting the readings over the network to the emulator, then that's an unfortunate incident and the resolution for such an incident is beyond the scope of the current work of Sensor Emulation.

For the remote-server scenario, the implementation is a straightforward one. The program **`SensorEmulationRemoteServer.c`** is run on the host and generates artificial sensor readings using random number generation method, and sending it to any connected client. This remote server could be anywhere and can be replaced any other server as well, as long as the pre-determined sensors reading pattern is followed while sending the respective sensor readings to any connected client. And the client running as part of the **`SensorEmulationClientServer.c`** compiled with pre-processor macro **`REMOTE_SERVER_READINGS`** will connect to the respective remote-server instance running as part of **`SensorEmulationRemoteServer.c`** based on the specific pre-determined sensor ports. And rest of the operations once the artificial readings are received remain the same as that's in the case of RAD specific real device sensor readings explained earlier, end-to-end.

Chapter 10
Evaluation

10.1 Time Delays Comparison

The evaluation involved comparing the timestamped sensor readings from the real and virtual phones. Before finalizing the correct time measuring API, APIs experimented with were, **`gettimeofday()`** and **`clock_gettime(CLOCK_MONOTONIC, . . .)`**. **`gettimeofday()`** was decided not to be used as it's marked deprecated. **`clock_gettime(CLOCK_MONOTONIC, . . .)`** is not used since it's based on the time elapsed since unspecified time in the past which can be different on the real device and on the emulator. So, the next reliable option was **`clock_gettime(CLOCK_REALTIME, . . .)`** and has been used for the evaluation. The obvious fluctuations in the measuring **`CLOCK_REALTIME`** is ignored at the cost of simplicity of logging the timestamps, as the **`CLOCK_REALTIME`** is the only available comparable time quantity between the real device and the emulator. To have the fluctuations compensated, a time-syncing application, ClockSync was installed on both the real device and the emulator to update the current time at any point time based on the internet based NTP server.

Based on the prolonged observation of three months of time and data collection, it has been noted that there's absolutely no loss of sensor readings being sent from the real-device to the emulator as long as the network connectivity stays in tact. This observation of three months is not a continuous day-and-night one but discrete one based on the times wherein there was any significant update in terms of the overall code. Only the latest observation and the numerical evaluation details is being discussed here as it accounts for the latest code with all the optimizations and updates that have made their way until the last day of that three months period from Sep, 2013 to Nov, 2013. This observation involved running 100+ top Android applications ranging from the basic sensor test applications that show the raw sensor readings, to the advanced 3D motion-sensor based Android games which are compatible with the emulator.

The list of those 100+ sensor-based applications tried are as under(in fact, they are 300+ in number, to be precise). All of these are taken from various Android websites in the internet in addition to Google Play/Android Market.

1. 101_in_1_games_hd.apk
2. 101_marbles.apk
3. 1MobileMarket_i.apk
4. 20130403134522 2b8796_Car_Racing_Game_1.1_9game_1.1.apk
5. 20131021_103741947.apk

6. 2xl_mx_offroad.apk
7. 3dbioballhd-1347872753270-apk.apk
8. 3d_bio_ball_hd.apk
9. 3d_drag_race.apk
10. 3d_rally_fever.apk
11. 4x4_offroad_racing.apk
12. absolute_rc_boat_sim.apk
13. absotruckinlutely.apk
14. Accelerometer.apk
15. Accelerometer Monitor.apk
16. ace_box_race.apk
17. ace_race_overdrive.apk
18. agracer.apk
19. air3-9_android.apk
20. airrace_skybox.apk
21. alpha_wheels_racing.apk
22. ambulance_rush.apk
23. AndroidSensorBox.apk
24. AndroSensor.apk
25. asphalt_5.apk
26. asphalt_6_adrenaline_hd.apk
27. asphalt_7_heat.apk
28. asphalt_8_airborne.apk
29. asphalt_moto.apk
30. atticlab.ScaryMazedGame.v1.20.apk
31. atv_madness.apk
32. badayer_racing.apk
33. Balance Ball 3D v.1.1.apk
34. bike_race.apk
35. blaz3d.apk
36. boomboom_racing.apk
37. breitling_reno_air_races.apk
38. Bubble_1.9.3.apk
39. burning_tires.apk
40. burning_wheels_3d_racing.apk
41. busted.apk
42. car_club_tuning_storm.apk
43. carmageddon.apk
44. car_race.apk
45. cars_and_guns_3d.apk
46. championship_jet_ski_2013.apk
47. championship_karting_2012.apk
48. championship_motorbikes_2013.apk

49. championship_racing_2013.apk
50. championship_rally_2012.apk
51. chundos_turbo.apk
52. city_cars_racer.apk
53. colin_mcrae_rally_hd.apk
54. com-dmw-stuntmarblespro-4-170986.myapp.apk
55. om.infraredpixel.drop.v1.4.5.apk
56. com-ratsquare-SpaceHell3D-1353898786120-apk.apk
57. corridor_fly.apk
58. crazxquad.apk
59. crazxracing.apk
60. crazyboat.apk
61. crazy_city_moto.apk
62. crazy_drive.apk
63. crazy_monster_truck.apk
64. crazy_monster_truck_escape.apk
65. crazyrush_volume_1.apk
66. crazy_taxi.apk
67. crazy_wheels_monster_trucks.apk
68. crc_pro_cycling.apk
69. csr_racing.apk
70. cubed_rally_redline.apk
71. cycling_2013.apk
72. deadly_moto_racing.apk
73. deathdrive.apk
74. death_rally_free.apk
75. death_rider.apk
76. death_tour.apk
77. dirt_karting.apk
78. dirt_road_trucker_3d.apk
79. doom_buggy.apk
80. drag_racing_3d.apk
81. drag_racing.apk
82. drag_racing_bike_edition.apk
83. draw_race_2.apk
84. driftkhana_freestyle_drift_app.apk
85. drift_mania_championship_2.apk
86. drift_mania_championship.apk
87. drift_mania_street_outlaws.apk
88. drive_with_zombies.apk
89. drop.1.0.apk
90. Drop It! (Ouch!.. Doh!) Pro New.apk
91. DropIt_v1.3_(stevebasu.net).apk

92. ducati_challenge.apk
93. dune_rider.apk
94. dust_offroad_racing.apk
95. duty_driver.apk
96. Earthquake Viewer.apk
97. Essential Tools - Lite.apk
98. extreme_biking_3d.apk
99. extreme_demolition.apk
100. extreme_formula.apk
101. extreme_racing_racing_moto.apk
102. f1_ultimate.apk
103. falling_marbles.apk
104. fast_five_the_movie_official_game_hd.apk
105. fast_furious_6_the_game.apk
106. fast_outlaw_asphalt_surfers (1).apk
107. fast_outlaw_asphalt_surfers.apk
108. fast_track_racers.apk
109. fatal_derby.apk
110. fennec-24.0.multi.android-arm.apk
111. fire_forget_the_final_assault (1).apk
112. fire_forget_the_final_assault.apk
113. fmx_iv_pro.apk
114. formula_racing_ultimate_drive.apk
115. formula_sprinty.apk
116. freestyle_dirt_bike.apk
117. furious_racing.apk
118. furious_wheel.apk
119. GamesApk.net-3DStuntMarblesPro.apk
120. GamesApk.net-Falldown3D.apk
121. g_bikes.apk
122. gecko-unsigned-unaligned.apk
123. Go Tetris.apk
124. grand_theft_auto_vice_city.apk
125. gravitire_3d.apk
126. gravity_tetris_3d.apk
127. ground_effect.apk
128. gt_racing_hyundai_edition.apk
129. GT Racing Motor Academy Free.apk
130. gt_racing_motor_academy_hd.apk
131. hallowheels.apk
132. hardcore_dirt_bike_2.apk
133. hardcore_dirt_bike.apk
134. hawthorne_park_thd.apk

135. hess_racer.apk
136. highway_rally.apk
137. highway_rider.apk
138. hot_mod_racer.apk
139. ion_racer.apk
140. island_racer.apk
141. jelly_racing.apk
142. jett_tailfin_racers.apk
143. jump_racer.apk
144. JumpyBall_3D.apk
145. kartrider_rush.apk
146. kinder_bueno_buggy_race_20.apk
147. konradgp.apk
148. kumho_tire_drive.apk
149. labyrinth_2.apk
150. labyrinth_lite.apk
151. lane_splitter.apk
152. lawn_mower_madness.apk
153. lego_pullback_racers.apk
154. lego_speedorz.apk
155. leviton_racers_hd.apk
156. link_237_racer.apk
157. Lucky Patcher v3.6.5.apk
158. mad_cop_car_race_and_drift.apk
159. Mad Race.apk
160. mad_skills_motocross.apk
161. marble_maze.apk
162. marble_run0.6c.apk
163. math_blaster_hyperblast_2.apk
164. megastunt_mayhem.apk
165. microworld_racing_3d.apk
166. minicar_champion_circuit_race.apk
167. mini_motor_racing.apk
168. mm_com.bogatov.tetris.apk
169. mm_com.thatWeirdKid.RaquetBall2D.apk
170. mole_kart.apk
171. monster_truck_destruction.apk
172. monster_wheels_offroad.apk
173. moto_gp_2012.apk
174. motogp_3d_super_bike_racing (1).apk
175. motogp_3d_super_bike_racing.apk
176. motoheroz.apk
177. moto_locos.apk

178. muscle_run.apk
179. MySensors.apk
180. need_for_drift.apk
181. need_for_speed_hot_pursuis.apk
182. need_for_speed_most_wanted.apk
183. need_for_speed_shift.apk
184. neon_city.apk
185. night_ride.apk
186. nissan_juke_nismo_challenge.apk
187. nos_car_speedrace.apk
188. official_speedway_gp_2013.apk
189. offroad_nation_pro.apk
190. pipe_glider.apk
191. pipe_rider.apk
192. pirate_wings.apk
193. piratewing_zhermq4i.apk
194. play.4.3.11.apk
195. plunk.apk
196. pocket_rally.apk
197. pocket_trucks.apk
198. pro_tennis_2013.apk
199. protoxide_death_race.apk
200. pure_drift.apk
201. Quick_Boot_4.1.apk
202. race_2.apk
203. race_horses_champions.apk
204. race_illegal_high_speed_3d.apk
205. race_n_chase_3d_car_racing.apk
206. race_of_champions.apk
207. race_rally_3d_car_racing.apk
208. racer_xt.apk
209. race_stunt_fight.apk
210. racing_glider.apk
211. racing_legends.apk
212. racing_moto.apk
213. radical_tube.apk
214. radio_ball_3d.apk
215. rage_truck.apk
216. raging_thunder_2.apk
217. rally_the_world_the_game.apk
218. rc_mini_racers.apk
219. real_moto_hd.apk
220. real_racing_2.apk

221. real_racing_3.apk
222. reckless_2.apk
223. reckless_getaway.apk
224. reckless_racing.apk
225. red_bull_ar_reloaded.apk
226. red_bull_kart_fighter_3.apk
227. red_bull_x_fighters_2012.apk
228. redline_rush.apk
229. red_wing_ikaro_racing.apk
230. renault_trucks_racing.apk
231. repulze.apk
232. return_zero.apk
233. re_volt_classic.apk
234. ricky_carmichaels_motocross.apk
235. riptide_gp2.apk
236. riptide_gp.apk
237. romanian_racing.apk
238. satans_zombies.apk
239. SC3D v1.1.3 apkmania.com.apk
240. Sensor List.apk
241. skatingirlz.apk
242. ski_challenge.apk
243. skyball.apk
244. skyriders_complete.apk
245. slingshot_racing.apk
246. smash_cops_heat.apk
247. smooth_3d.apk
248. snowbike_racing.apk
249. snuggle_truck.apk
250. socketserver-0.2.apk
251. sonic_sega_all_stars_racing.apk
252. space_rings_3d.apk
253. speed_car.apk
254. SpeedCar.apk
255. speedcarii.apk
256. speed_forge_3d.apk
257. speedmoto2.apk
258. speedmoto.apk
259. speed_night_2.apk
260. speedway_grand_prix_2011.apk
261. speedx_3d.apk
262. sports_car_challenge.apk
263. starbounder.apk

264. Steady_Compass_-_1.3.3(freemaza.in).apk
265. sunkist_speedway.apk
266. supasupacross.apk
267. supercrosspro.apk
268. supermotoracing_1381991920376.apk
269. suspect_the_run.apk
270. swift_adventure.apk
271. switch.apk
272. tangya.apk
273. tgear_test_track.apk
274. the_final_escape.apk
275. the_jump_escape_the_city.apk
276. tiki_kart_3d.apk
277. tiny_little_racing_2.apk
278. tires_of_fury_monster_truck_racing.apk
279. top_gear_stunt_school_revolution.apk
280. toyota_86_ar.apk
281. t_racer_hd.apk
282. tractor_farm_driver.apk
283. tractor_more_farm_driving.apk
284. transporters.apk
285. tube_racer_3d.apk
286. turbofly_3d.apk
    a. turbo_racing_league.apk
287. ultimate_3w.apk
288. ultimate_motocross_2.apk
289. ultimate_motocross.apk
290. ultra4_offroad_racing.apk
291. urbanchaser_speed_3d_racing.apk
292. warp_dash.apk
293. wave_blazer.apk
294. whacksy_taxi.apk
295. wheel_rush.apk
296. wrc_shakedown_edition.apk
297. xdrag.apk
298. yamaha_ttx_revolution.apk
299. YAMM_v1.1.apk
300. zaxxon_escape.apk
301. zombie_driver_thd.apk
302. zombie_gp.apk
303. zombie_highway.apk

Out of the above 100+ applications(300+, in fact!) tried in a span of three months, the

sensor based applications that **work smoothly** and hence **used** in the evaluation were : My Sensors, Championship Motorbikes 2013, Light-O-meter, Micro Metal Detector, ExionFly 3D HD, Proximity Launcher, Night Compass, Master Spirit Level, Android Sensor Box, and Sensor Test. The number of sensor events for each of these sensors differ based on how many readings was generated/sensed by the real device sensor in a span of approximately 10 minutes of evaluation. With that, Accelerometer and Gyroscope readings were recorded for just above 1000 sensor events and all other sensors' readings having been recorded for approximately 100 sensor events.

It's highly tedious to keep track of and mention here the list of specific problems that each of the 100+(300+, actually) sensor-based applications including the advanced 3D games experienced in a span of three months period while running them the emulator since there were cases of running the same application more than at least 20 times with several variations in terms of the graphics support, ARM translation, Hardware Acceleration so on and so forth whose details can be found under "Graphics" subsection in the next chapter. As if this was not enough, there were occasions of testing the same 3D racing game version with all the available copies of the game tailored for different handset sizes like VGA, SVGA, etc as it appeared any graphics issue would be solved with the lower graphics versions of the same game. Also, different copies of the same game were tried to avoid "Downloading game data" as most of the recent 3D games demanded WiFi connection which sometimes worked and some other times didn't as described in the "Wifi necessity" section in the next chapter. There were a whole lot of other issues mentioned in the next chapter, amidst which keeping track of extensive list of which version of what application faced which issue was simply impossible.

Fig. 10.1 depicts the graphical comparison of the time delays applicable for the sensor readings in reaching the emulator from the real Android device.

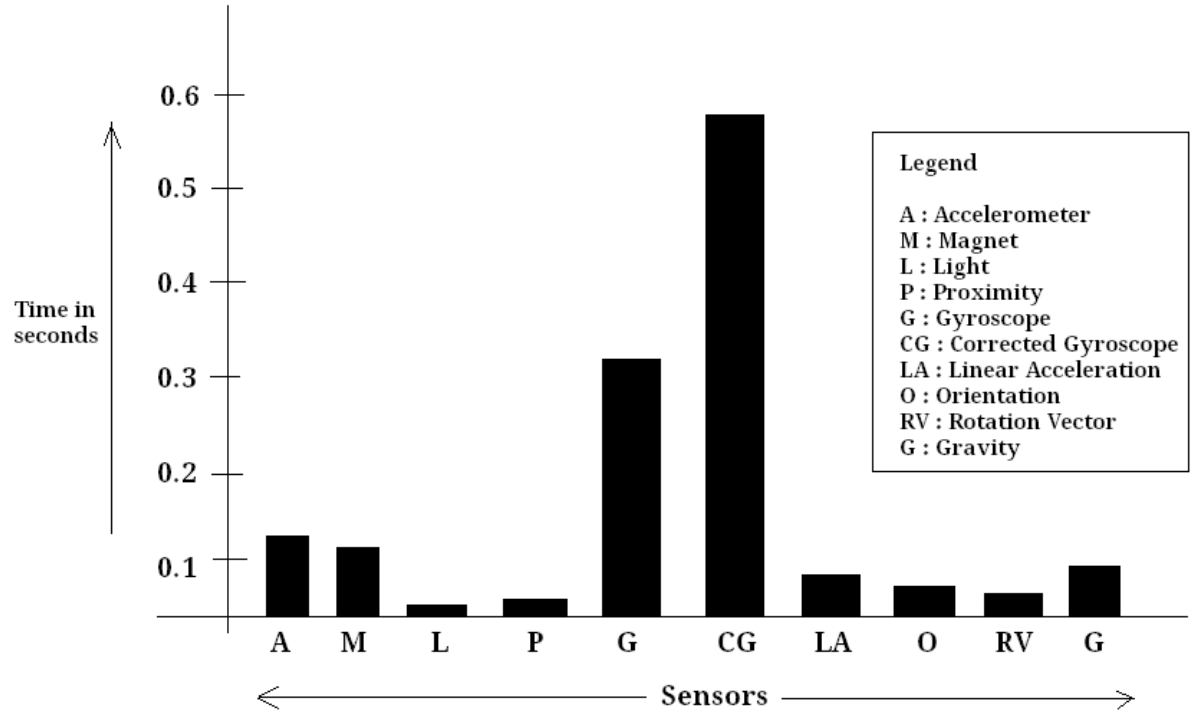


Fig. 10.1 Pictorial comparison of the average time delays applicable for the sensor readings in reaching the Virtual Android Device(VAD) from the Real Android Device(RAD)

As it can be seen, the average time delays of Corrected Gyroscope, Gyroscope, and Accelerometer are the three highest average time delays. The reason for such a difference is that Gyroscope and Accelerometer in that order are having higher sampling frequencies than the other real hardware sensors. As a result, there will be relatively high influx of sensor readings from these two sensors onto their respective network buffer as each sensor has a dedicated server dealing with its own socket-connection and hence their own network buffer as well. To put things in perspective, there were approximately at least 1000 independent and consecutive sensor events for accelerometer and gyroscope(Corrected gyroscope) for which there were approximately only 100 sensor events of other slower sensors. So, the sensor readings of accelerometer and gyroscope spend a little longer in their respective network buffers than what sensor readings of other sensors with relatively lower sampling frequency would spend in their network buffers. And as far as the Corrected Gyroscope is concerned, it's just a virtual sensor using the same Gyroscope readings with the highest sampling frequency among all the sensors being emulated and hence all that has been said for gyroscope holds good for Corrected Gyroscope as well.

Having become clear so as to why there is a noticeable difference between the average time delays of the emulated sensors, it's time to look at the impact due to the relatively

high average time delays. Well, it's quite obvious that there will be a tiny delay that's inevitable due to the transmission of readings over the network. But, how tiny is this delay depends on the factors mentioned already. And all these delays are the ones that are got from the latest code which has been fine-tuned and optimized in every possible manner though under 0.6 seconds of delay should be tolerable given that the entire Sensor Emulation is happening over the network at full possible speed.

Going even more **extreme** in an attempt to reduce the delay, one can reduce or remove the delay in the client-servers between send-receive calls -- this would make the Sensor Emulation code to be hogging CPU all the time and the whole Android-x86 UI on the emulator will be sluggish; not acceptable, or one can reduce the sampling frequency of high sampling frequency sensors on the real device so that the number of sensor readings of accelerometer and gyroscope drop and become on par with the other relatively slower sensors and thus having the almost same very small delay of say under 0.15 seconds -- this would kill the purpose of making Sensor Emulation to be lossless and accurate while making it speedy; not acceptable, and so on.

In short, there will definitely be **slight tradeoff** between speed and accuracy of sensor readings in reaching the destination(VAD) from source(RAD). And in the current work, accuracy is being favored first and then the speed. However, this tradeoff doesn't affect the sensor-based applications to the extent of hampering their sensor-based responsiveness. This has been verified by running the emulator compatible sensor-based applications using two 3D sensor-based games wherein one uses Accelerometer and the other uses Gyroscope. And the demo videos clearly supports this claim of the sensor-based applications working smoothly with utmost responsiveness as seen on the real device. So, the current work is very much usable in real-world scenario without any serious concerns. More information about the usability factor of the current work is described in the "Usability Factor" chapter ahead.

Below Table 10.1 gives the numerical comparison of the average time delays applicable for the sensor readings in reaching the Virtual Android Device(VAD) from the Real Android Device(RAD). These are the numerical data based on which the bar graph comparison in Fig. 10.1 is depicted.

Statistics → Sensors	**Lowest delay(in nanoseconds)**	**Highest delay(in nanoseconds)**	**Average delay(in nanoseconds)**	**Number of readings missed during the transmission**	**Number of readings with delays above the delay thresho**

↓				from the real device to the emulator	ld of 4 seconds
Accelerometer	2954496	432915456	123727873	0	0
Magnetic	2304	843893248	111025584	0	0
Light	134656	91592704	9595756	0	0
Proximity	4751616	81309440	12776519	0	0
Gyroscope	23935744	1375095808	327739803	14	0
Corrected Gyroscope	5789696	1124702464	592972183	0	0
Linear Acceleration-n	20992	624702208	69012239	0	0
Orientation	8448	622296832	54785776	0	0
Rotation Vector	112384	347361792	46375939	0	0
Gravity	8704	727843328	96040132	0	0

Table 10.1 Numerical comparison of the average time delays applicable for the sensor readings in reaching the Virtual Android Device(VAD) from the Real Android Device(RAD)

There are couple of interesting things in the above table that need attention.

1. The 14 missing readings reported for the gyroscope are not really missing but they were at the very end of the transmission of readings from the RAD to VAD and hence failed to reach the VAD and hence were not logged and thus are reported to be missing.
2. The sensor readings of each sensor are collected with all other sensors enabled at the same time, except for the gyroscope and accelerometer whose readings were collected with only two of them enabled at the same time to measure them accurately considering their high-frequencies compared to other sensors.

In the above readings, the lowest and the highest timestamps differ significantly for some of the sensors due to the delays considered end-to-end. This includes the initial time taken for the network initialization necessary for the initial network packet to reach from the real device to the virtual device as well as the time taken for the very last few network packets in reaching the virtual phone from the real phone. Especially for the very last few packets(sensor readings), when the end-to-end network connection is interrupted by pressing Ctrl+C at the terminal when there is a need, there will be a slight delay for those specific sensor readings to reach the virtual phone from the real phone as apparently their end-to-end network connectivity is interrupted due to the break in the connecting link which is the intermediate program facilitating the interaction between the real and virtual phones, and the intermediate program is the one which receives(handles) the Ctrl+C at the terminal.

In addition, to all the delays described above, there will be a slight increase in the network bandwidth usage as and when additional sensors are enabled end-to-end that is the real and virtual devices. These additional sensors require independent network connections for their sensor readings to reach from the real to virtual phones over the network. More and more sensor readings start to occupy the network bandwidth, there will be a gradual but negligible increase in the time taken for the sensor readings to reach from the real phone to the virtual phone.

Based on all the above valid reasons for the possible delays for the sensor readings to reach from the real phone to virtual phone, it should be logical to see the relatively glaring difference in the lowest and highest time taken for the sensor readings of a particular sensor.

If at all there is a need for a uniform distribution of time taken for the sensor readings, ideally, the first few and last few sensor readings should be ignored when measuring the overall time taken for a particular sensor's readings to reach from the real phone to the virtual phone. Thereby, the minor spikes in the overall time taken will be absent. And the above numerical table will have reasonable differences for the timestamps of sensor readings for each of the sensor. However, such a precision wasn't thought as important during the evaluation of the current work of Sensor Emulation and hence

one can see the apparent differences in the lowest and highest time taken for the sensor readings.

10.2 Comprehensive Benchmarking

For getting deeper insight of the end-to-end Sensor Emulation involving the comparisons of every minute detail of the real hardware sensors with virtualized sensors, a neatly written and immensely useful sensor-test application from Android Market, SensMark was used both on the real and virtual devices by launching them manually almost at the same time in parallel. Below are the complete and exhaustive logs for the sake of completeness of evaluation.

Here are the descriptions of what the detailed logs mean as told over the e-mail by one of the lead developers of the SensMark project.

1. ***Number of Events:*** the number of measured values during a benchmark (~10 sec), the more the better
2. ***Time Consistency:*** depends on the minimum time between 2 measured succeeded values during initialization and benchmark. the lesser the better. but in comparison with each other, meaning Time Consistency should be equal during initialization and benchmark, indicating if the sensors only recognizes changes when moved or at specific time interval. the more equal the better.
3. ***Standard Deviation:*** The lesser the standard deviation (by Donald E. Knuth) the better. For devices that should not be moved during benchmark.
4. ***Resolution:*** Minimum difference between two succeeding values, just like the Resolution given by the sensor, only that our value is measured and true, and the given one probably not.
5. ***Gravity:*** Only at Accelerometer and Gravity Sensors: How close is the average z-axis value during benchmark to 9.81m/s^2 with a 3% offset due to devices without flat rear/back covers.
6. ***Light & Proximity***
   Number of Events: same as above
   Deviation timepitch: Does not depend on results of initialization and describes the Standard deviation over all time steps between two succeeding measured values.
7. ***Resolution:*** same as above.
8. ***Differing values:*** How many different values can this sensor measure. Proximity often only have 0 and 5. The Galaxy S2 default Light sensor only measures 10, 100, 1000, 10000 and 16000 lux, but with Cyanogenmod it shows all values between 0 and I don't know how it can get.

10.2.1 Complete Logs Location

Real device - http://sensmark.info/sensor-benchmarks/?shw=u&u=2857
Virtual device - http://sensmark.info/sensor-benchmarks/?shw=u&u=2858

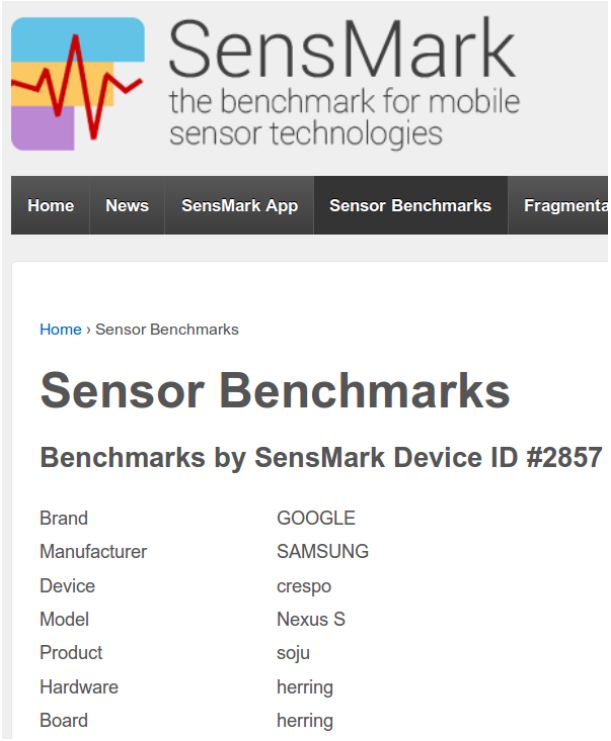
Following sections have their content based on the logs at these webpages. The following sections serve as a ready-reference and also a persistent one if in case, the online benchmark logs are lost due to some reasons.

The real device had 4.3 while virtual device had 4.3.1. However, there are no much differences between these two Jelly Bean versions of Android Release. Here is a snapshot of the details of these two releases from the corresponding wiki page.

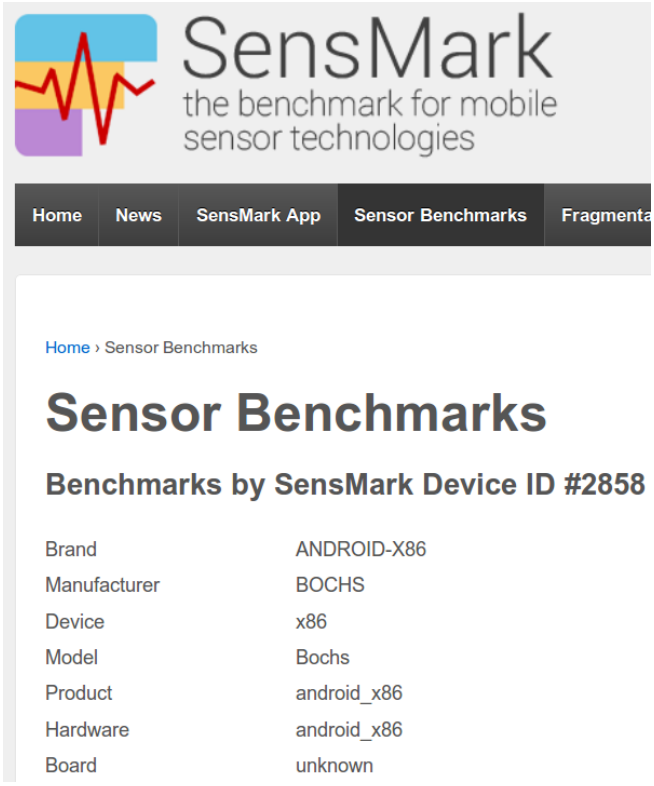
Version	Release date	Features
4.3	24 July 2013[111]	• Bluetooth low energy support.[112] • Bluetooth Audio/Video Remote Control Profile (AVRCP) 1.3 support • OpenGL ES 3.0 support, allowing for improved game graphics[112] • Restricted access mode for new user profiles[112] • Filesystem write performance improvement by running fstrim command while device is idle[113] • Dial pad auto-complete in the Phone application[112] • Improvements to Photo Sphere[114] • Reworked camera UI, previously introduced on Google Play edition phones[115] • Addition of "App Ops", a fine-grained application permissions control system (hidden by default)[116] • 4K resolution support[117] • Many security enhancements, performance enhancements, and bug fixes[118] • System-level support for geofencing and Wi-Fi scanning APIs • Background Wi-Fi location still runs even when Wi-Fi is turned off • Developer logging and analyzing enhancements • Added support for five more languages • Improved digital rights management (DRM) APIs • Right-to-left (RTL) languages now supported[112] • Clock in the status bar disappears if clock is selected as lockscreen widget
4.3.1	3 October 2013[119]	• Bug fixes and small tweaks for the Nexus 7 LTE[120]

10.3 Devices Details

Real Device



Virtual Device



10.4 Comparing Benchmarking Points

This section compares and contrasts the points which is the numerical sum of number of sensor events, deviation timepitch, deviation values, and sensor resolution.

10.4.1 Accelerometer

Real Device

Accelerometer

KR3DM 3-axis Accelerometer by Hacked!! STMicroelectronics

Date	Android Version	Points Total	Points Deviation Timestep	Points Deviation Values	Points Resolution	Points Events	Points Gravity
2013-12-29 15:38:05.92 EST	4.3	12157	1832	3167	3236	331	3591

Note: THe **"Hacked!!"** word in the title is a remnant of the very initial set of experiments of Sensor Emulation and has really nothing to do with any hacking to achieve the needed sensor emulation.

Virtual Device

Accelerometer

Accelerometer Sensor Emulation!!!! by Columbia University - NYC - Thesis - Raghavan Santhanam

Date	Android Version	Points Total	Points Deviation Timestep	Points Deviation Values	Points Resolution	Points Events	Points Gravity
2013-12-29 15:38:06.486 EST	4.3.1	12485	1785	3167	3236	707	3590

10.4.2 Magnetometer

Real Device

Magnetic Field

AK8973 3-axis Magnetic field sensor by Asahi Kasei Microdevices

Date	Android Version	Points Total	Points Deviation Timestep	Points Deviation Values	Points Resolution	Points Events
2013-12-29 15:38:06.694 EST	4.3	6606	1826	2217	2245	318

Virtual Device

Magnetic Field

Magnetic sensor Emulation!!!! by Columbia University - NYC - Thesis - Raghavan Santhanam

Date	Android Version	Points Total	Points Deviation Timestep	Points Deviation Values	Points Resolution	Points Events
2013-12-29 15:38:08.429 EST	4.3.1	6269	1230	2195	2467	377

10.4.3 LIGHT

Real Device

Light Sensor

GP2A Light sensor by Sharp

Date	Android Version	Points Total	Points Deviation Timestep	Points Resolution	Points Differing Values
2013-12-29 16:27:01.129 EST	4.3	4633	2089	1892	652

Virtual Device

Light Sensor

Light sensor Emulation!!!! by Columbia University - NYC - Thesis - Raghavan Santhanam

Date	Android Version	Points Total	Points Deviation Timestep	Points Resolution	Points Differing Values
2013-12-29 16:27:00.450 EST	4.3.1	5224	1913	2711	600

10.4.4 PROXIMITY

Real Device

Proximity Sensor

GP2A Proximity sensor by Sharp

Date	Android Version	Points Total	Points Deviation Timestep	Points Resolution	Points Differing Values
2013-12-29 17:17:10.682 EST	4.3	1214	845	367	2

Virtual Device

Proximity Sensor

Proximity sensor Emulation!!!! by Columbia University - NYC - Thesis - Raghavan Santhanam

Date	Android Version	Points Total	Points Deviation Timestep	Points Resolution	Points Differing Values
2013-12-29 17:17:11.931 EST	4.3.1	1647	1278	367	2

10.4.5 Gyroscope

Real Device

K3G Gyroscope sensor by STMicroelectronics

Date	Android Version	Points Total	Points Deviation Timestep	Points Deviation Values	Points Resolution	Points Events
2013-12-29 15:38:09.587 EST	4.3	16697	2390	4023	4431	5853

Virtual Device

Gyroscope

Gyroscope sensor Emulation!!!! by Columbia University - NYC - Thesis - Raghavan Santhanam

Date	Android Version	Points Total	Points Deviation Timestep	Points Deviation Values	Points Resolution	Points Events
2013-12-29 15:38:09.564 EST	4.3.1	15875	3416	4015	4431	4013

10.4.6 Corrected Gyroscope

Real Device

Gyroscope

Corrected Gyroscope Sensor by Google Inc.

Date	Android Version	Points Total	Points Deviation Timestep	Points Deviation Values	Points Resolution	Points Events
2013-12-29 15:38:27.709 EST	4.3	23700	2408	3778	9750	7764

Virtual Device

Corrected Gyroscope Sensor Emulation!!!! by Google Inc.

Date	Android Version	Points Total	Points Deviation Timestep	Points Deviation Values	Points Resolution	Points Events
2013-12-29 15:38:17.365 EST	4.3.1	18521	2311	3786	8662	3762

Note: The vendor of emulated Corrected Gyroscope was modified from ***"Columbia University - NYC - Thesis - Raghavan Santhanam"*** to ***"Google Inc."*** because, the SensMark looks out for the exact vendor string to be "Google Inc." in order to say whether a sensor is virtual or not. And when the Corrected Gyroscope is not treated as a virtual sensor due to its type being same as that of the default Gyroscope, its data will be treated as redundant as Gyroscope would have already uploaded with its type. Hence the Corrected Gyroscope's sensor data would never appear in the webpage showing the analysis of the sensor data collected and uploaded to http://sensmark.info.
As a matter of fact, the SensMark application's code isn't open sourced. So, the only option left is to just ***visualize*** the Sensor characteristics that are expected by SensMark for a sensor to be virtual so that Gyroscope and Corrected Gyroscope sensor data are treated as separate ones allowing both of them to be part of the analysis.

Thus, the issue of missing Corrected Gyroscope sensor data in the sensmark.info webpage was solved with just pure ***visualization*** and ***expertise*** in programming with emphasis on memory handling(for strings, to be specific).

10.4.7 Orientation

Real Device

Orientation

Orientation Sensor by Google Inc.

Date	Android Version	Points Total	Points Deviation Timestep	Points Deviation Values	Points Resolution	Points Events
2013-12-30 01:54:08.110 EST	4.3	14554	3433	3074	7567	480

Virtual Device

Orientation

Orientation Sensor Emulation!!!! by Columbia University - NYC - Thesis - Raghavan Santhanam

Date	Android Version	Points Total	Points Deviation Timestep	Points Deviation Values	Points Resolution	Points Events

2013-12-30 01:54:11.904 EST	4.3.1	11627	2025	3388	5868	346

10.4.8 Gravity

Real Device

Gravity

Gravity Sensor by Google Inc.

Date	Android Version	Points Total	Points Deviation Timestep	Points Deviation Values	Points Resolution	Points Events	Points Gravity
2013-12-30 19:51:57.825 EST	4.3	19940	3688	4983	7194	473	3602

Virtual Device

Gravity

Gravity Sensor Emulation!!!! by Columbia University - NYC - Thesis - Raghavan Santhanam

Date	Android Version	Points Total	Points Deviation Timestep	Points Deviation Values	Points Resolution	Points Events	Points Gravity
2013-12-30 19:52:00.222 EST	4.3.1	18643	2424	4973	7194	450	3602

10.4.9 Linear Acceleration

Real Device

Linear Acceleration

Linear Acceleration Sensor by Google Inc.

Date	Android Version	Points Total	Points Deviation Timestep	Points Deviation Values	Points Resolution	Points Events
2013-12-30 19:46:43.989 EST	4.3	14370	3643	3142	7113	472

Virtual Device

Linear Acceleration

Linear Acceleration Sensor Emulation!!!! by Columbia University - NYC - Thesis - Raghavan Santhanam

Date	Android Version	Points Total	Points Deviation Timestep	Points Deviation Values	Points Resolution	Points Events
2013-12-30 19:46:46.368 EST	4.3.1	12387	2394	2466	7113	414

10.4.10 Rotation Vector

Real Device

Rotation Vector

Rotation Vector Sensor by Google Inc.

Date	Android Version	Points Total	Points Deviation Timestep	Points Deviation Values	Points Resolution	Points Events
2013-12-30 19:40:00.87 EST	4.3	17733	3631	5064	8562	476

Virtual Device

Rotation Vector

Rotation Vector Sensor Emulation!!!! by Columbia University - NYC - Thesis - Raghavan Santhanam

Date	Android Version	Points Total	Points Deviation Timestep	Points Deviation Values	Points Resolution	Points Events
2013-12-30 19:40:02.272 EST	4.3.1	16722	2649	5061	8562	450

10.5 Detailed Benchmarking Comparisons

This section is a back-to-back detailed collection and comparison of the Sensor characteristics and behavior for the real phone and for the virtual phone which bear relation with the earlier section on the points benchmarking analysis in an one-to-one manner. So, by clicking on the above entries in the respective web pages already listed for real and virtual devices, the following statistical analysis of sensor data can be seen.

There will be apparent differences in the graphs obtained for the performances of the sensors on the real and virtual phones, in spite of the fact the exact same copy of SensMark application was installed and used on both real and virtual phones. The main reason is the time delay which is an aggregate of the minute difference in time between the manual launch of the SensMark application on real and virtual phones, the network delay for the sensor readings to reach from the real phone to virtual phone, the minute time delay induced by the processing of the incoming sensor data within the virtual phone, i.e., virtualized Android-x86. Hence, these differences are inevitable but negligible as far as the overall tolerable accuracy is concerned.

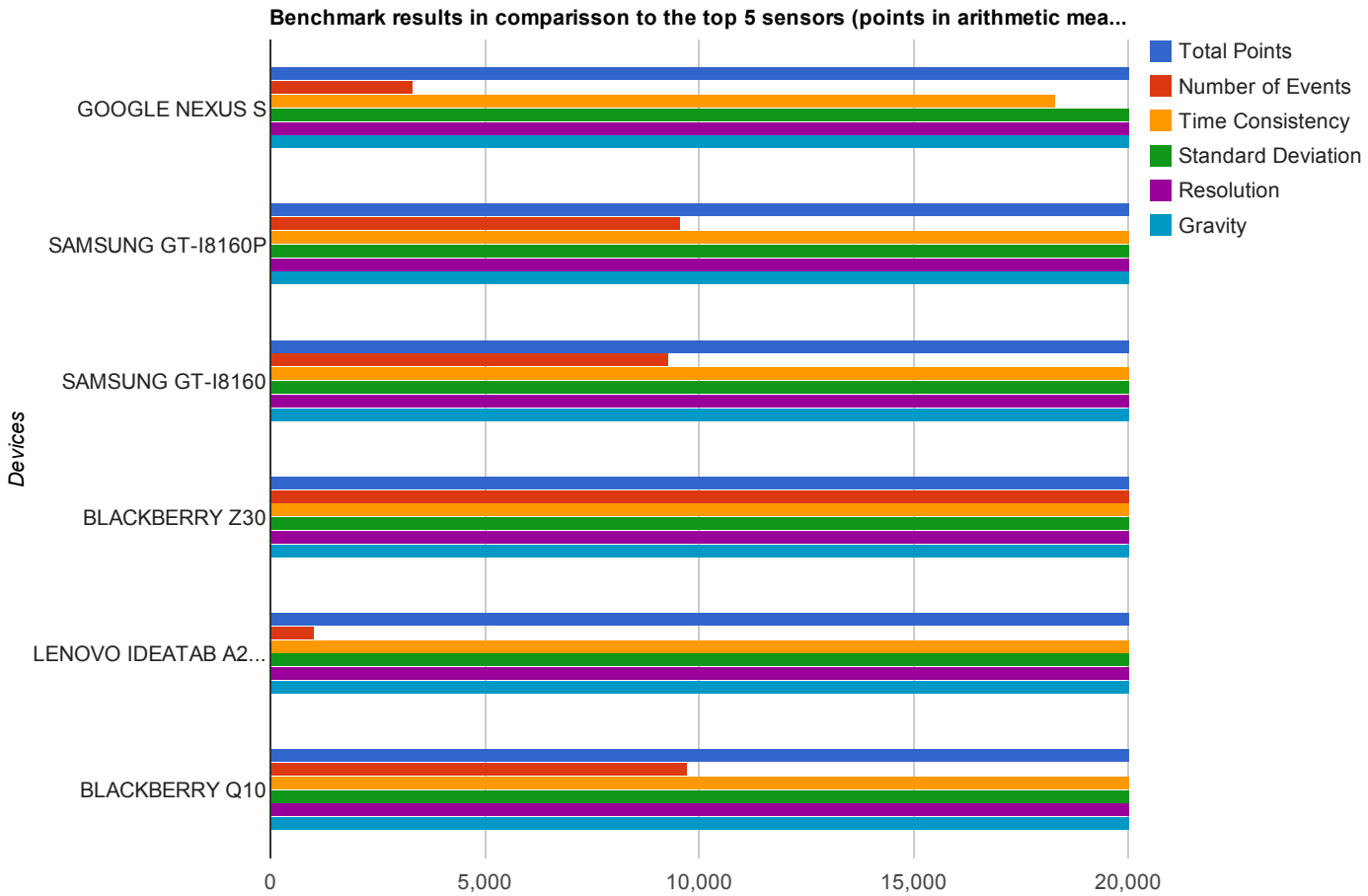
The following statistical analysis includes two separate graphical plots for each of the sensors on the real and virtual phones. First one is of the initialization of sensors with

some sample sensor data and second one is of the benchmarking of the actual sensor data from the real hardware sensors on real and virtual phones. As mentioned already, the two graphical plots differ slightly due to the inherent delays in the real phone to frame a network buffer consisting of the real hardware sensor readings and send it across the network, the delay caused by the network overhead for the sensor readings to reach from the real to virtual phone, the delay imposed by the inherent virtual limit on the rate at which the emulator virtualizing Android-x86, can process the incoming network data, and finally, the delay due to the internal processing of the received network data within the Sensors HAL module of the virtual phone. So, these delays are integral to the Sensor Emulation but tolerable ones.

Sensor Benchmarks

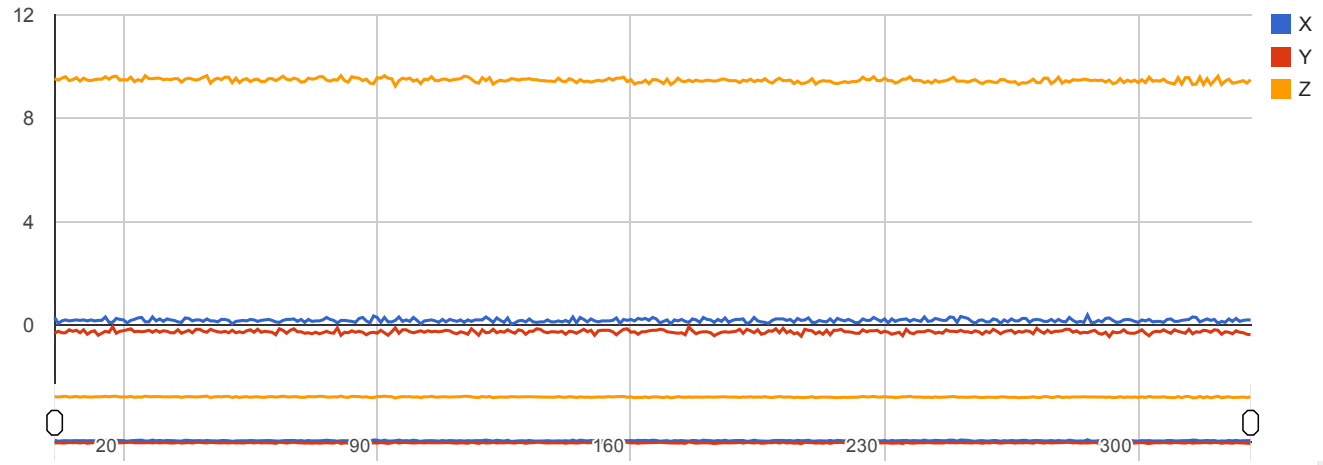
Accelerometer: KR3DM 3-axis Accelerometer by Hacked!! STMicroelectronics

Benchmark



Points

Total	12157
Events	331
Deviation Timepitch	1832
Deviation Values	3167
Resolution	3236
Gravity	3591



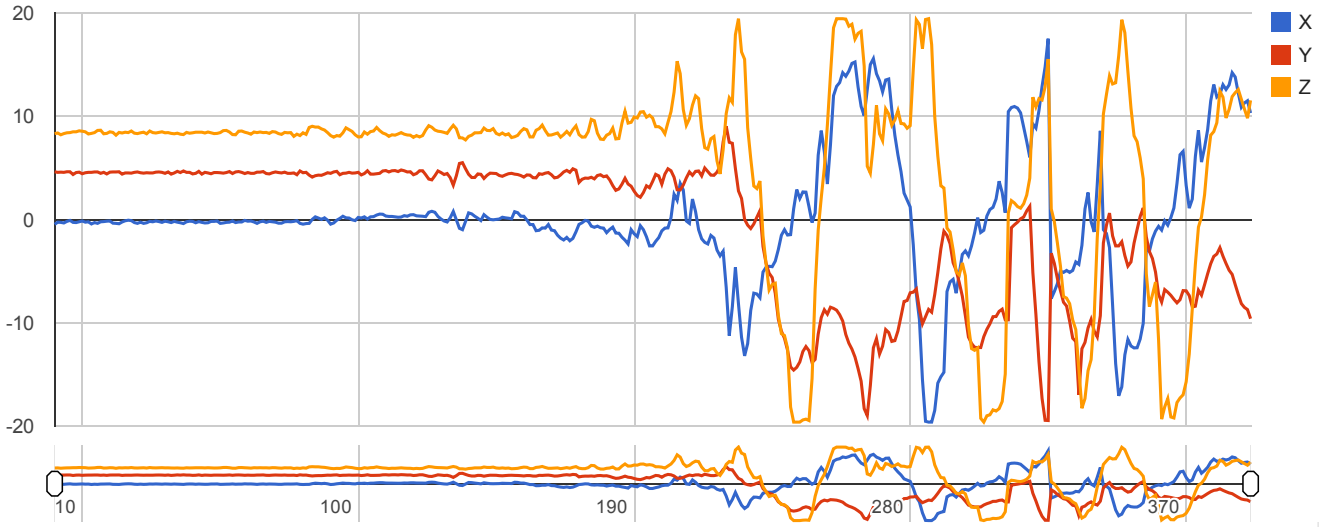
Download all values as JSON array

Number of events	331
Timepitch (minimum)	19095000 µs
Timepitch (arithmetic mean)	24281006.0606061 µs
Timepitch (standard deviation)	40403454.5990557 µs
Minimum Resolution X	0.019153595 m/s²
Min. Resolution Y	0.019153595 m/s²
Min. Resolution Z	0.019153595 m/s²
Minimum X	0.019153614 m/s²
Mean X	0.173597704889281 m/s²
Maximum X	0.38307226 m/s²
Minimum Y	-0.45968673 m/s²
Mean Y	-0.267571931023972 m/s²
Maximum Y	-0.076614454 m/s²
Minimum Z	9.251195 m/s²
Mean Z	9.47502097910625 m/s²
Maximum Z	9.653421 m/s²
Standard Deviation X	0.0637146600280529 m/s²
Standard Deviation Y	0.0630660436415817 m/s²
Standard Deviation Z	0.0770003221419178 m/s²
Android Version	4.3
App Version	1.0
App Version Code	9
Battery	72%
Charging	0

SensMark results based on all executed initializations and benchmarks with this specific sensor

Minimum Timepitch	17990000 µs
Minimum Resolution X	0.019153595 m/s²
Min. Resolution Y	0.019153595 m/s²
Min. Resolution Z	0.019153595 m/s²
Minimum X	-19.6133 m/s²
Maximum X	17.525557 m/s²
Minimum Y	-19.460072 m/s²
Maximum Y	9.040505 m/s²
Minimum Z	-19.6133 m/s²
Maximum Z	19.460072 m/s²

Initialization



Download all values as JSON array

Number of events	391
Timepitch (minimum)	17990000 µs
Timepitch (arithmetic mean)	27164705.1282051 µs
Timepitch (standard deviation)	59611069.570246 µs
Minimum Resolution X	0.019153595 m/s²
Min. Resolution Y	0.019153595 m/s²
Min. Resolution Z	0.019153595 m/s²
Minimum X	-19.6133 m/s²
Maximum X	17.525557 m/s²
Minimum Y	-19.460072 m/s²
Maximum Y	9.040505 m/s²
Minimum Z	-19.6133 m/s²
Maximum Z	19.460072 m/s²

About Device

Technical sensor specifications by device

Maximum Range	19.6133 m/s²	
Minimum Delay	20000 µs	The minimum delay allowed between two events in microsecond or zero if this sensor only returns a value when the data it's measuring changes.
Power	0.23 mA	The power in mA used by this sensor while in use.
Resolution	1 m/s²	
Version	1	

Note: In our research we found out that the specifications above do not have to be true.

Device

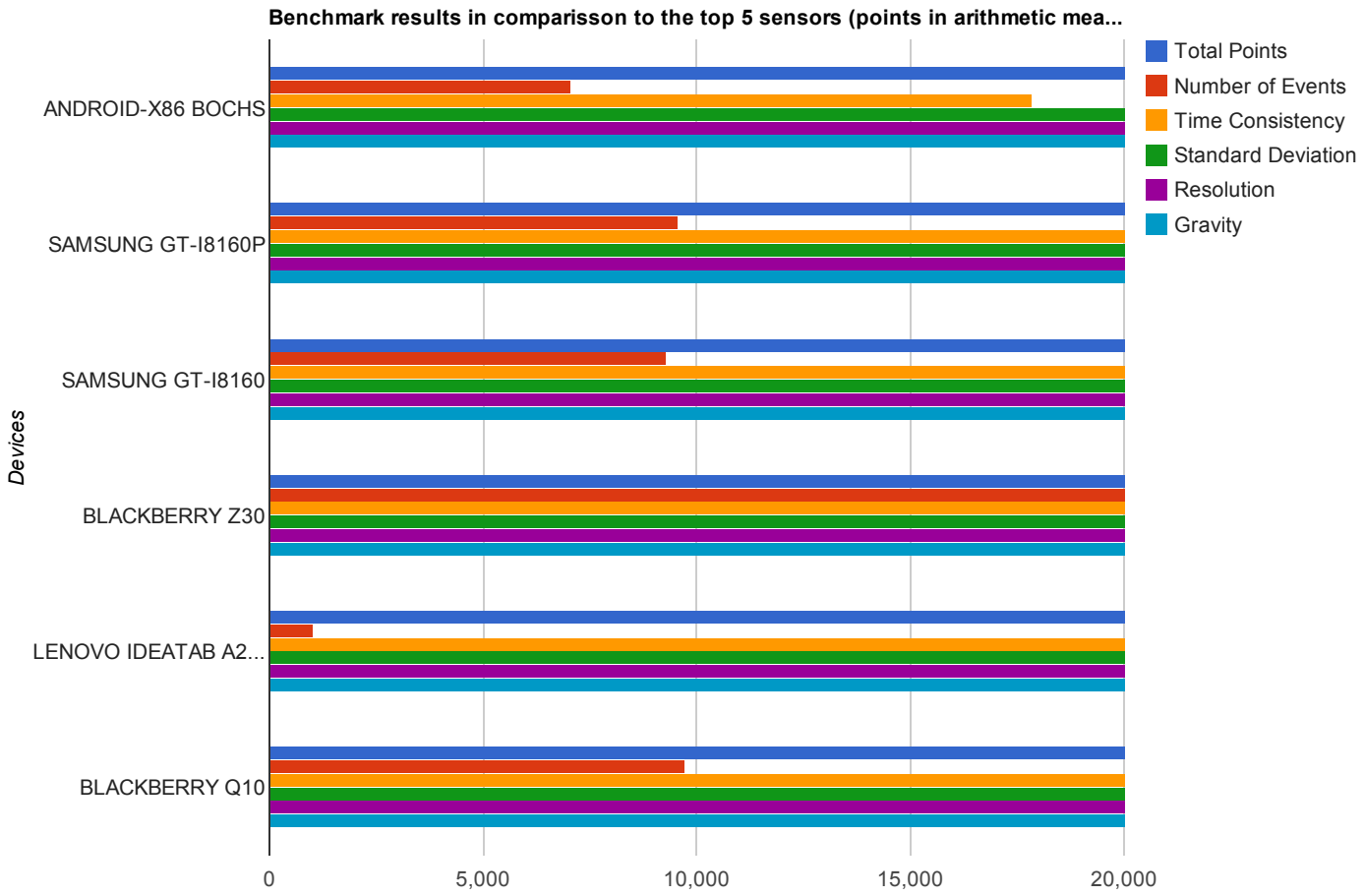
Brand	Google
Manufacturer	Samsung
Device	crespo
Model	Nexus S
Product	soju
Hardware	herring
Board	herring

↑

Sensor Benchmarks

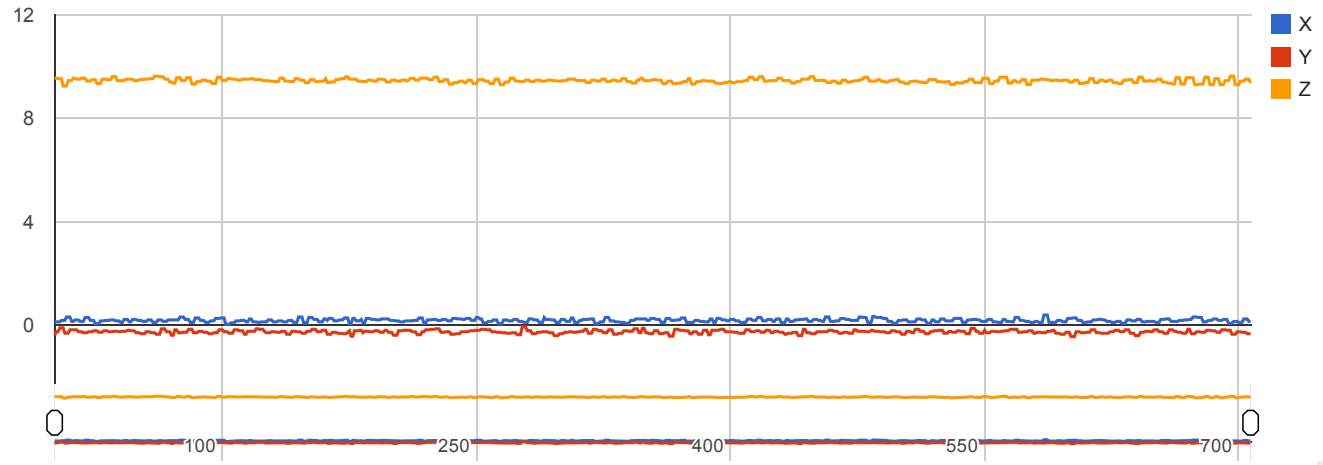
Accelerometer: Accelerometer Sensor Emulation!!!! by Columbia University - NYC - Thesis - Raghavan Santhanam

Benchmark



Points

Total	12485
Events	707
Deviation Timepitch	1785
Deviation Values	3167
Resolution	3236
Gravity	3590



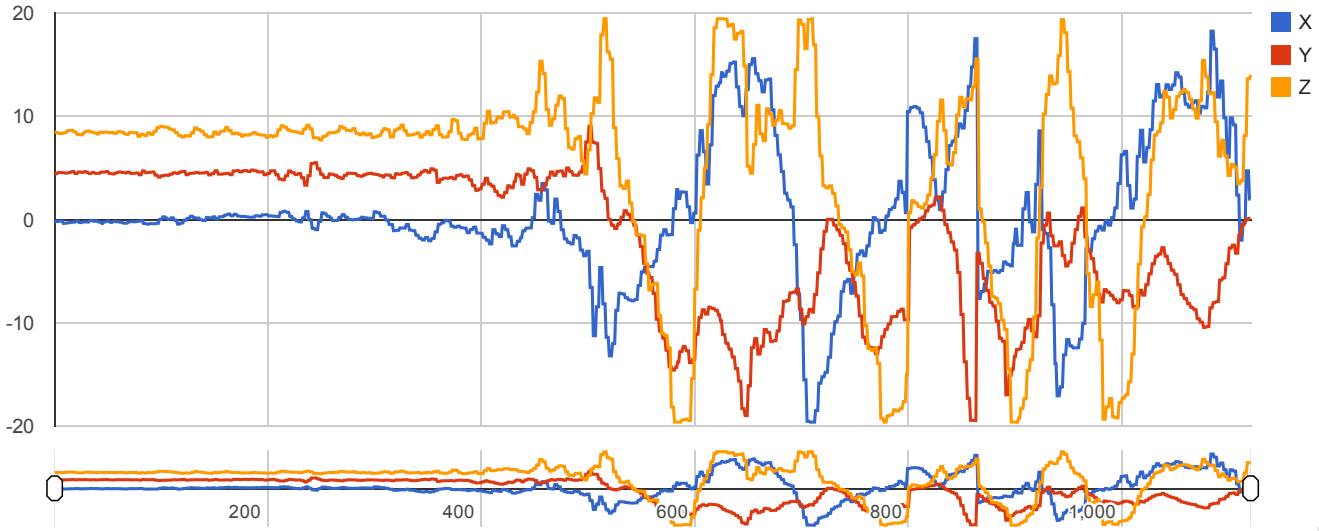
Download all values as JSON array

Number of events	707
Timepitch (minimum)	1030316 µs
Timepitch (arithmetic mean)	7932245.87535411 µs
Timepitch (standard deviation)	47662232.8685073 µs
Minimum Resolution X	0.01915361 m/s²
Min. Resolution Y	0.019153595 m/s²
Min. Resolution Z	0.019153595 m/s²
Minimum X	0.019153614 m/s²
Mean X	0.172978531679894 m/s²
Maximum X	0.38307226 m/s²
Minimum Y	-0.45968673 m/s²
Mean Y	-0.270344992728095 m/s²
Maximum Y	-0.076614454 m/s²
Minimum Z	9.251195 m/s²
Mean Z	9.45985356753079 m/s²
Maximum Z	9.634268 m/s²
Standard Deviation X	0.0651540174118856 m/s²
Standard Deviation Y	0.0631879314289808 m/s²
Standard Deviation Z	0.0749032701862655 m/s²
Android Version	4.3.1
App Version	1.0
App Version Code	9
Battery	100%
Charging	0

SensMark results based on all executed initializations and benchmarks with this specific sensor

Minimum Timepitch	1030316 µs
Minimum Resolution X	0.019153595 m/s²
Min. Resolution Y	0.019153595 m/s²
Min. Resolution Z	0.019153595 m/s²
Minimum X	-19.6133 m/s²
Maximum X	18.253393 m/s²
Minimum Y	-19.460072 m/s²
Maximum Y	9.040505 m/s²
Minimum Z	-19.6133 m/s²
Maximum Z	19.460072 m/s²

Initialization



Download all values as JSON array

Number of events	1120
Timepitch (minimum)	1044881 µs
Timepitch (arithmetic mean)	8716990.5549598 µs
Timepitch (standard deviation)	62387261.12377 µs
Minimum Resolution X	0.019153595 m/s²
Min. Resolution Y	0.019153595 m/s²
Min. Resolution Z	0.019153595 m/s²
Minimum X	-19.6133 m/s²
Maximum X	18.253393 m/s²
Minimum Y	-19.460072 m/s²
Maximum Y	9.040505 m/s²
Minimum Z	-19.6133 m/s²
Maximum Z	19.460072 m/s²

About Device

Technical sensor specifications by device

Maximum Range	19.6133 m/s²	
Minimum Delay	20000 µs	The minimum delay allowed between two events in microsecond or zero if this sensor only returns a value when the data it's measuring changes.
Power	3 mA	The power in mA used by this sensor while in use.
Resolution	1 m/s²	
Version	1	

Note: In our research we found out that the specifications above do not have to be true.

Device

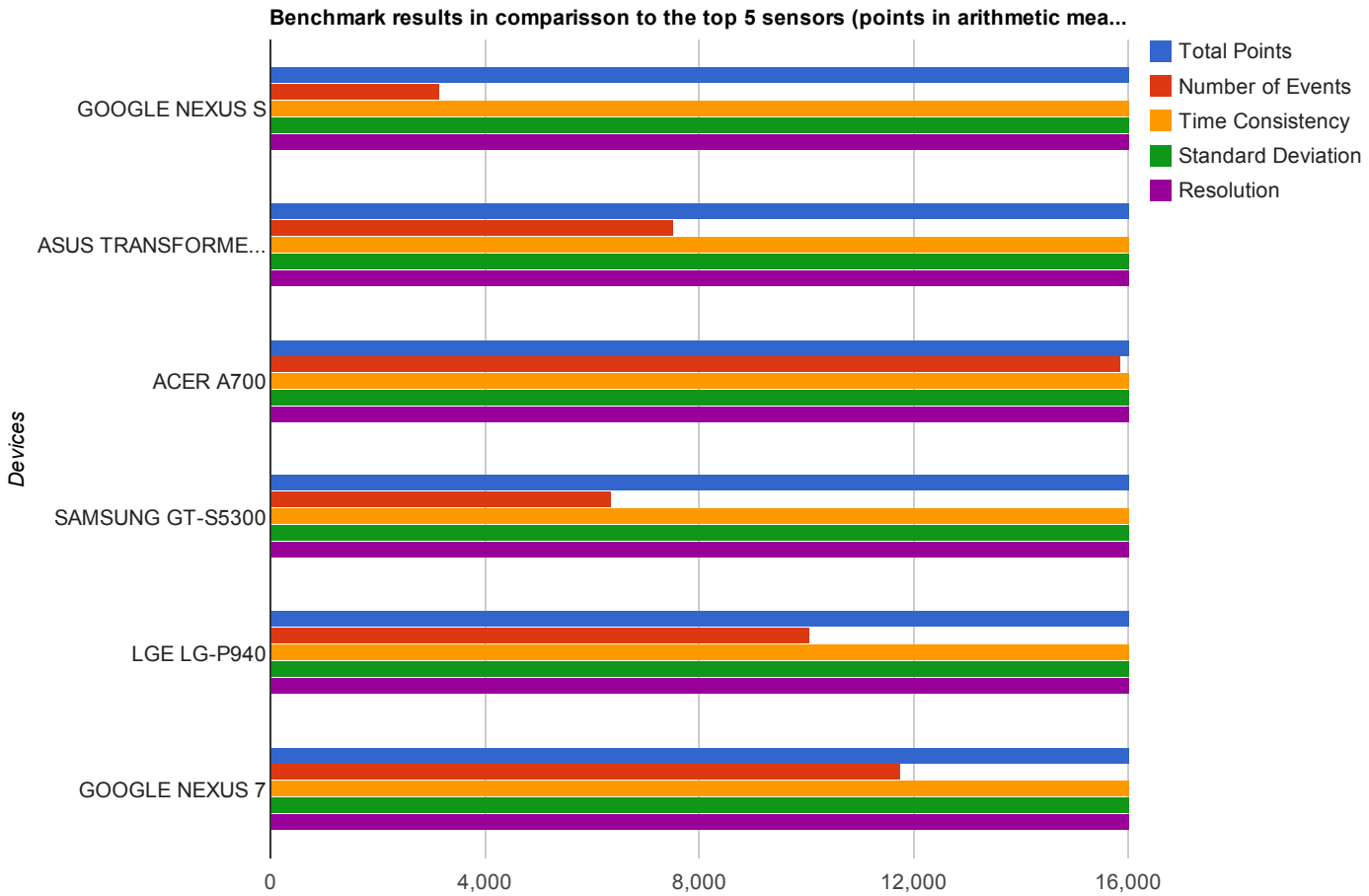
Brand	Android-x86
Manufacturer	Bochs
Device	x86
Model	Bochs
Product	android_x86
Hardware	android_x86
Board	unknown

↑

Sensor Benchmarks

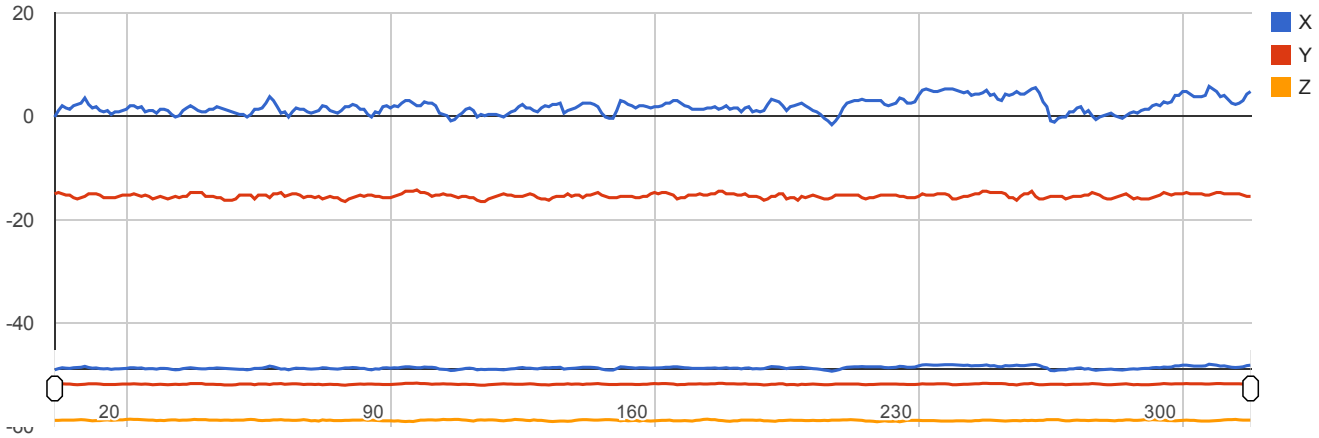
Magnetic Field: AK8973 3-axis Magnetic field sensor by Asahi Kasei Microdevices

Benchmark



Points

Total	6606
Events	318
Deviation Timepitch	1826
Deviation Values	2217
Resolution	2245



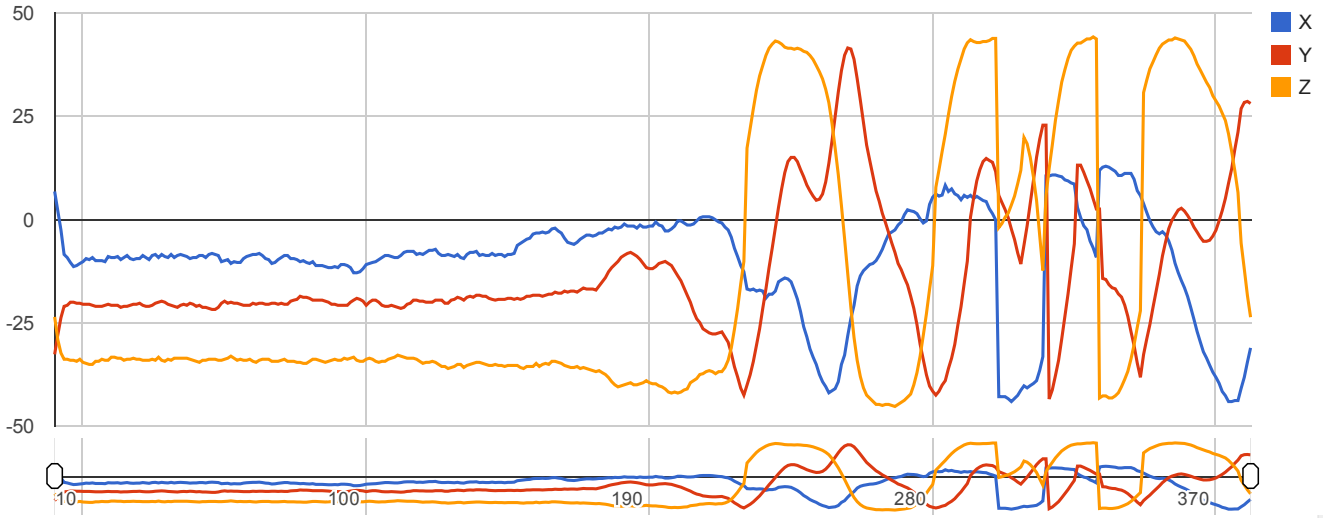
Download all values as JSON array

Number of events	318
Timepitch (minimum)	18932000 µs
Timepitch (arithmetic mean)	24698649.8422713 µs
Timepitch (standard deviation)	41054031.2759508 µs
Minimum Resolution X	0.1875 µT
Min. Resolution Y	0.25 µT
Min. Resolution Z	0.25 µT
Minimum X	-1.625 µT
Mean X	1.9433962264151 µT
Maximum X	5.8125 µT
Minimum Y	-16.5 µT
Mean Y	-15.3993710691824 µT
Maximum Y	-14.25 µT
Minimum Z	-55.375 µT
Mean Z	-53.8671383647799 µT
Maximum Z	-52.625 µT
Standard Deviation X	1.47648151237647 µT
Standard Deviation Y	0.414275522902881 µT
Standard Deviation Z	0.524946582566972 µT
Android Version	4.3
App Version	1.0
App Version Code	9
Battery	72%
Charging	0

SensMark results based on all executed initializations and benchmarks with this specific sensor

Minimum Timepitch	16986000 µs
Minimum Resolution X	0.1875 µT
Min. Resolution Y	0.1875 µT
Min. Resolution Z	0.1875 µT
Minimum X	-44.0625 µT
Maximum X	12.9375 µT
Minimum Y	-43.4375 µT
Maximum Y	41.5625 µT
Minimum Z	-55.375 µT
Maximum Z	44.25 µT

Initialization



Download all values as JSON array

Number of events	381
Timepitch (minimum)	16986000 µs
Timepitch (arithmetic mean)	27511194.7368421 µs
Timepitch (standard deviation)	59967726.9472331 µs
Minimum Resolution X	0.1875 µT
Min. Resolution Y	0.1875 µT
Min. Resolution Z	0.1875 µT
Minimum X	-44.0625 µT
Maximum X	12.9375 µT
Minimum Y	-43.4375 µT
Maximum Y	41.5625 µT
Minimum Z	-45.25 µT
Maximum Z	44.25 µT

About Device

Technical sensor specifications by device

Maximum Range	2000 µT	
Minimum Delay	16667 µs	The minimum delay allowed between two events in microsecond or zero if this sensor only returns a value when the data it's measuring changes.
Power	6.8 mA	The power in mA used by this sensor while in use.
Resolution	1 µT	
Version	1	

Note: In our research we found out that the specifications above do not have to be true.

Device

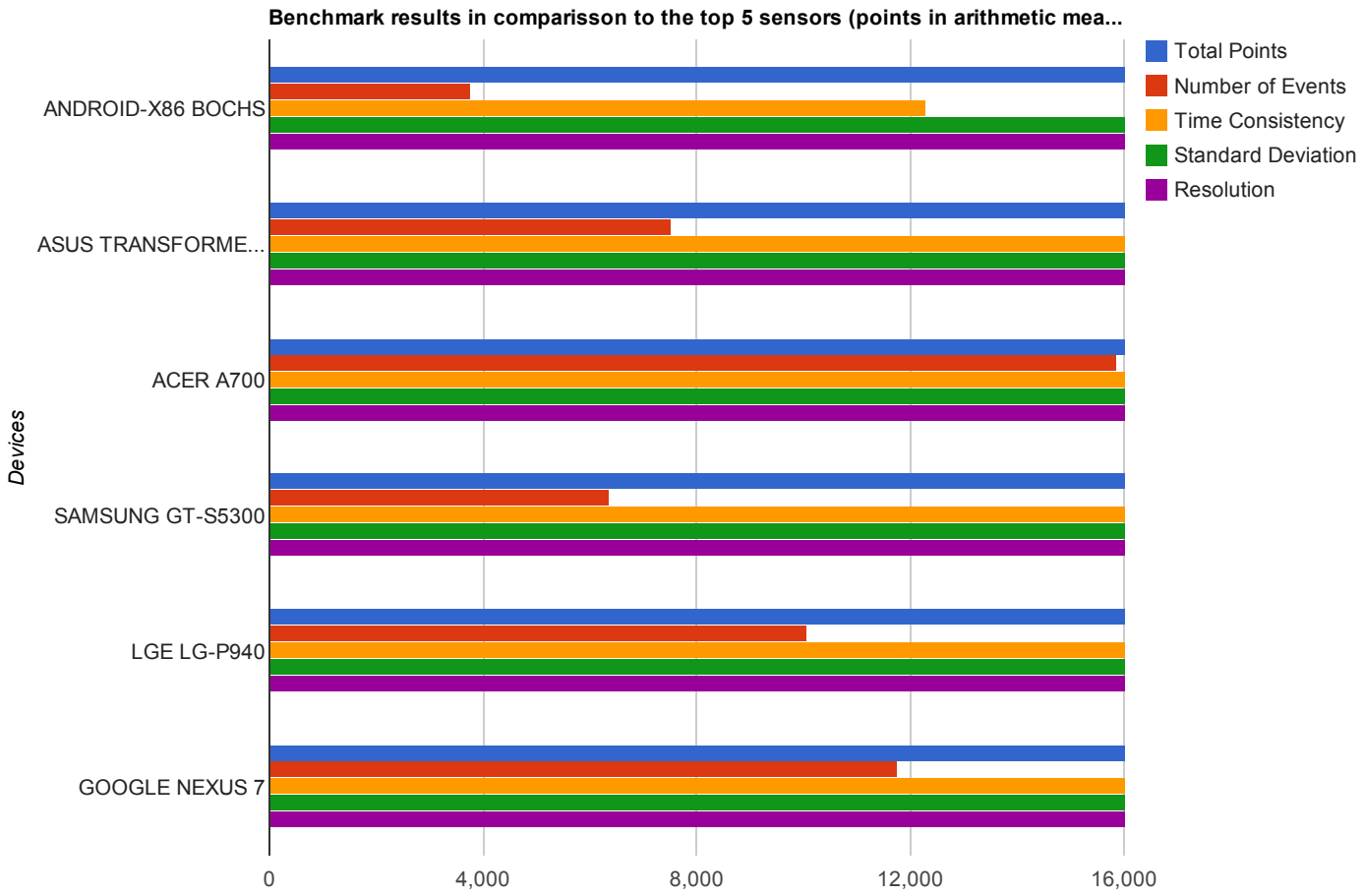
Brand	Google
Manufacturer	Samsung
Device	crespo
Model	Nexus S
Product	soju
Hardware	herring
Board	herring

↑

Sensor Benchmarks

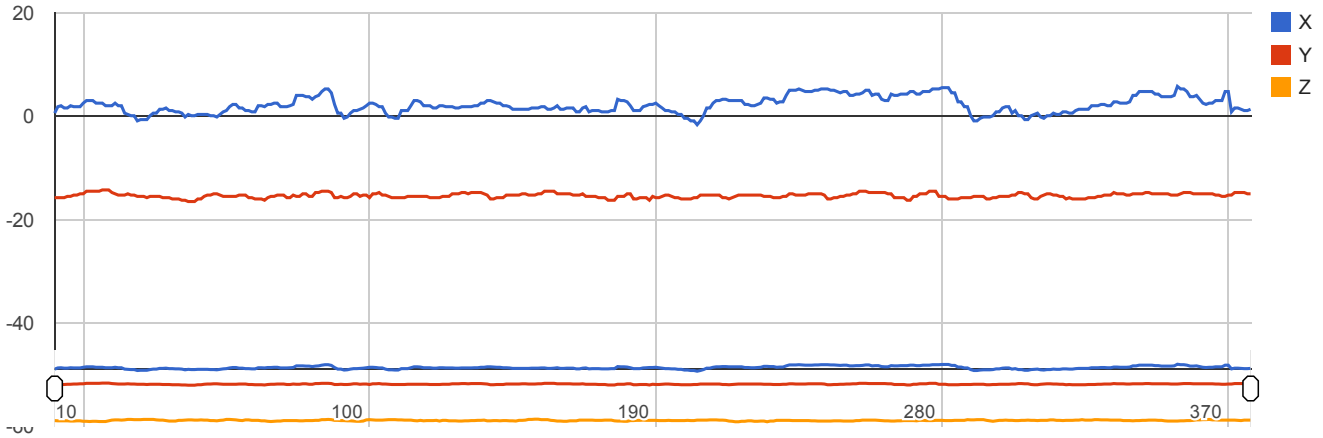
Magnetic Field: Magnetic sensor Emulation!!!! by Columbia University - NYC - Thesis - Raghavan Santhanam

Benchmark



Points

Total	6269
Events	377
Deviation Timepitch	1230
Deviation Values	2195
Resolution	2467



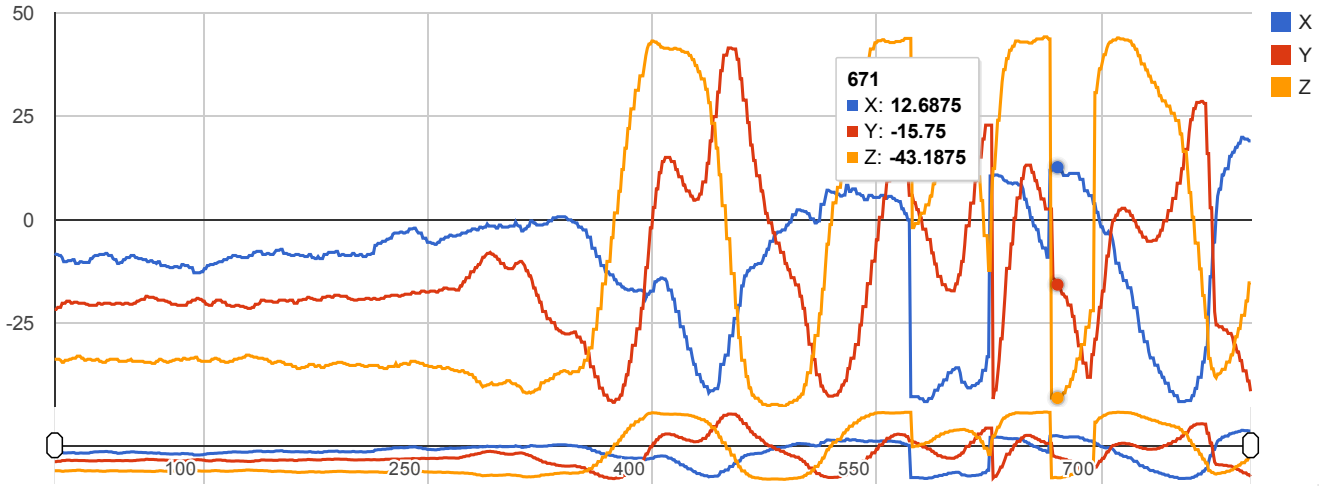
Download all values as JSON array

Number of events	377
Timepitch (minimum)	1068090 µs
Timepitch (arithmetic mean)	26520893.1223404 µs
Timepitch (standard deviation)	213634021.028299 µs
Minimum Resolution X	0.1875 µT
Min. Resolution Y	0.25 µT
Min. Resolution Z	0.25 µT
Minimum X	-1.625 µT
Mean X	2.22579575596817 µT
Maximum X	5.8125 µT
Minimum Y	-16.5 µT
Mean Y	-15.3554376657825 µT
Maximum Y	-14.25 µT
Minimum Z	-55.375 µT
Mean Z	-53.9711538461539 µT
Maximum Z	-52.625 µT
Standard Deviation X	1.62315598013038 µT
Standard Deviation Y	0.44249459903287 µT
Standard Deviation Z	0.529268169661988 µT
Android Version	4.3.1
App Version	1.0
App Version Code	9
Battery	100%
Charging	0

SensMark results based on all executed initializations and benchmarks with this specific sensor

Minimum Timepitch	1055755 µs
Minimum Resolution X	0.1875 µT
Min. Resolution Y	0.0625 µT
Min. Resolution Z	0.1875 µT
Minimum X	-44.0625 µT
Maximum X	19.9375 µT
Minimum Y	-44.25 µT
Maximum Y	41.5625 µT
Minimum Z	-55.375 µT
Maximum Z	44.25 µT

Initialization



Download all values as JSON array

Number of events	800
Timepitch (minimum)	1055755 µs
Timepitch (arithmetic mean)	12498430.7722153 µs
Timepitch (standard deviation)	56170854.6310429 µs
Minimum Resolution X	0.1875 µT
Min. Resolution Y	0.0625 µT
Min. Resolution Z	0.1875 µT
Minimum X	-44.0625 µT
Maximum X	19.9375 µT
Minimum Y	-44.25 µT
Maximum Y	41.5625 µT
Minimum Z	-45.25 µT
Maximum Z	44.25 µT

About Device

Technical sensor specifications by device

Maximum Range	2000 µT	
Minimum Delay	16667 µs	The minimum delay allowed between two events in microsecond or zero if this sensor only returns a value when the data it's measuring changes.
Power	6.8 mA	The power in mA used by this sensor while in use.
Resolution	1 µT	
Version	1	

Note: In our research we found out that the specifications above do not have to be true.

Device

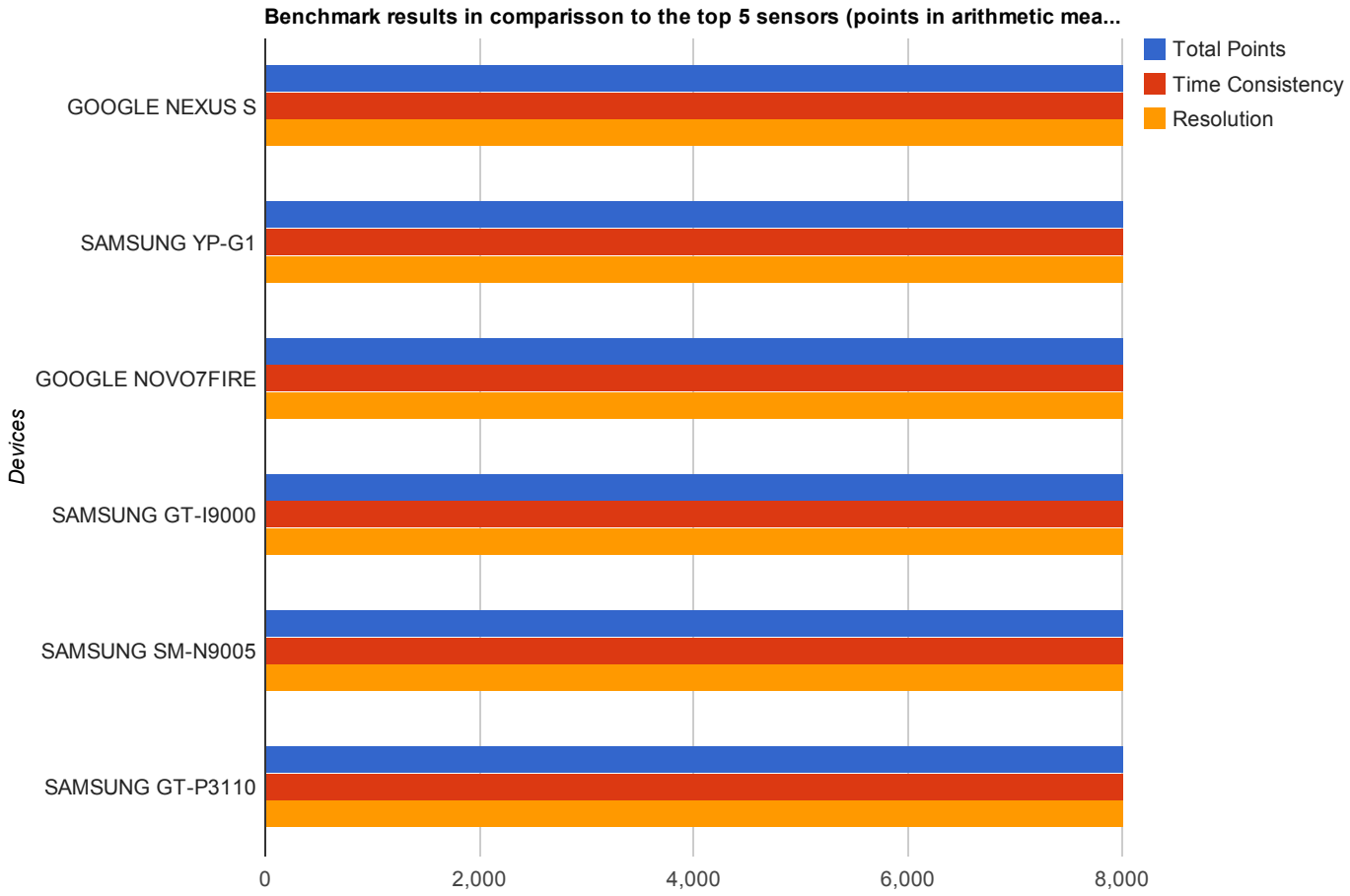
Brand	Android-x86
Manufacturer	Bochs
Device	x86
Model	Bochs
Product	android_x86
Hardware	android_x86
Board	unknown

↑

Sensor Benchmarks

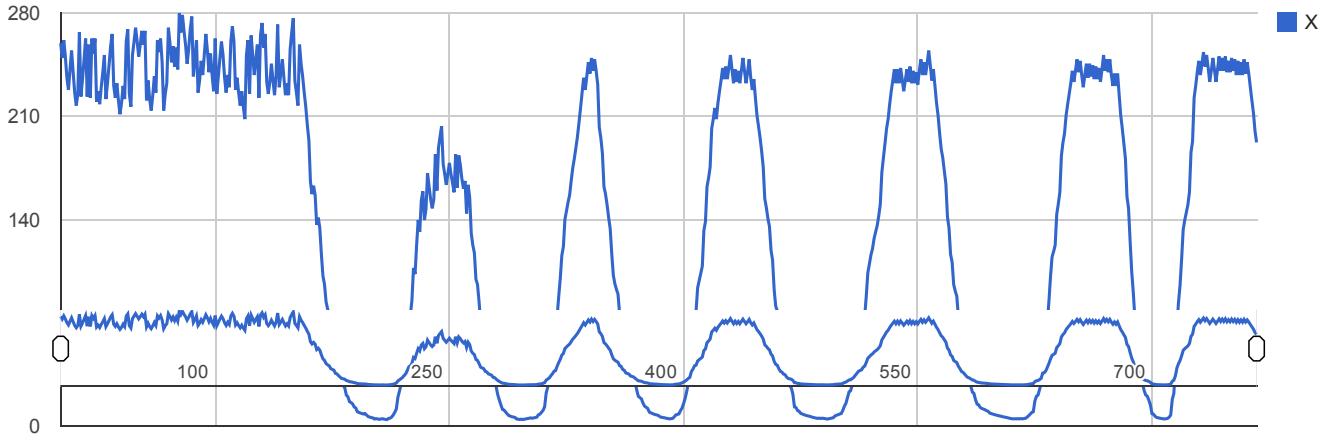
Light: GP2A Light sensor by Sharp

Benchmark



Points

Total	4633
Events	767
Deviation Timepitch	2089
Resolution	1892
Differing Values	652

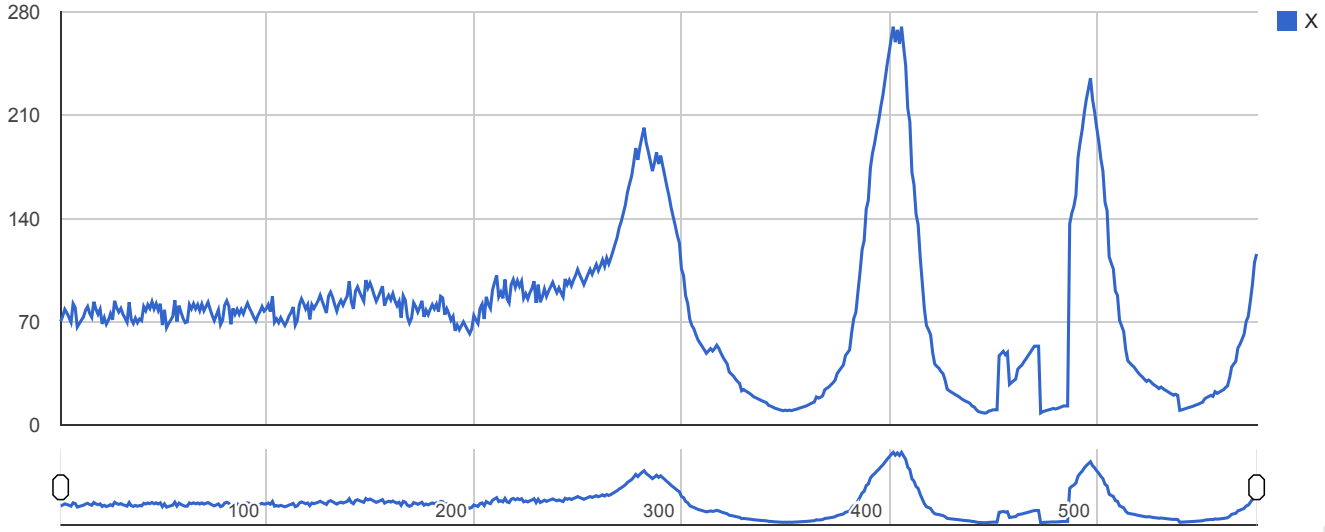


Number of events	767
Timepitch (minimum)	2166000 µs
Timepitch (arithmetic mean)	13385364.229765 µs
Timepitch (standard deviation)	18246983.518185 µs
Minimum Resolution X	0.14110899 lux
Minimum X	4.3975453 lux
Mean X	135.912088386705 lux
Maximum X	279.8769 lux
Standard Deviation X	99.5388267974499 lux
Android Version	4.3
App Version	1.0
App Version Code	9
Battery	54%
Charging	0

SensMark results based on all executed initializations and benchmarks with this specific sensor

Minimum Timepitch	2166000 µs
Minimum Resolution X	0.14110899 lux
Minimum X	4.3975453 lux
Maximum X	279.8769 lux

Initialization



Download all values as JSON array

Number of events	577
Timepitch (minimum)	4194000 µs
Timepitch (arithmetic mean)	18376986.1111111 µs
Timepitch (standard deviation)	50225284.1315726 µs
Minimum Resolution X	0.26020622 lux
Minimum X	8.109105 lux
Maximum X	270.10687 lux

About Device

Technical sensor specifications by device

Maximum Range	3626657.8 lux	
Minimum Delay	0 µs	The minimum delay allowed between two events in microsecond or zero if this sensor only returns a value when the data it's measuring changes.
Power	0.75 mA	The power in mA used by this sensor while in use.
Resolution	1 lux	
Version	1	

Note: In our research we found out that the specifications above do not have to be true.

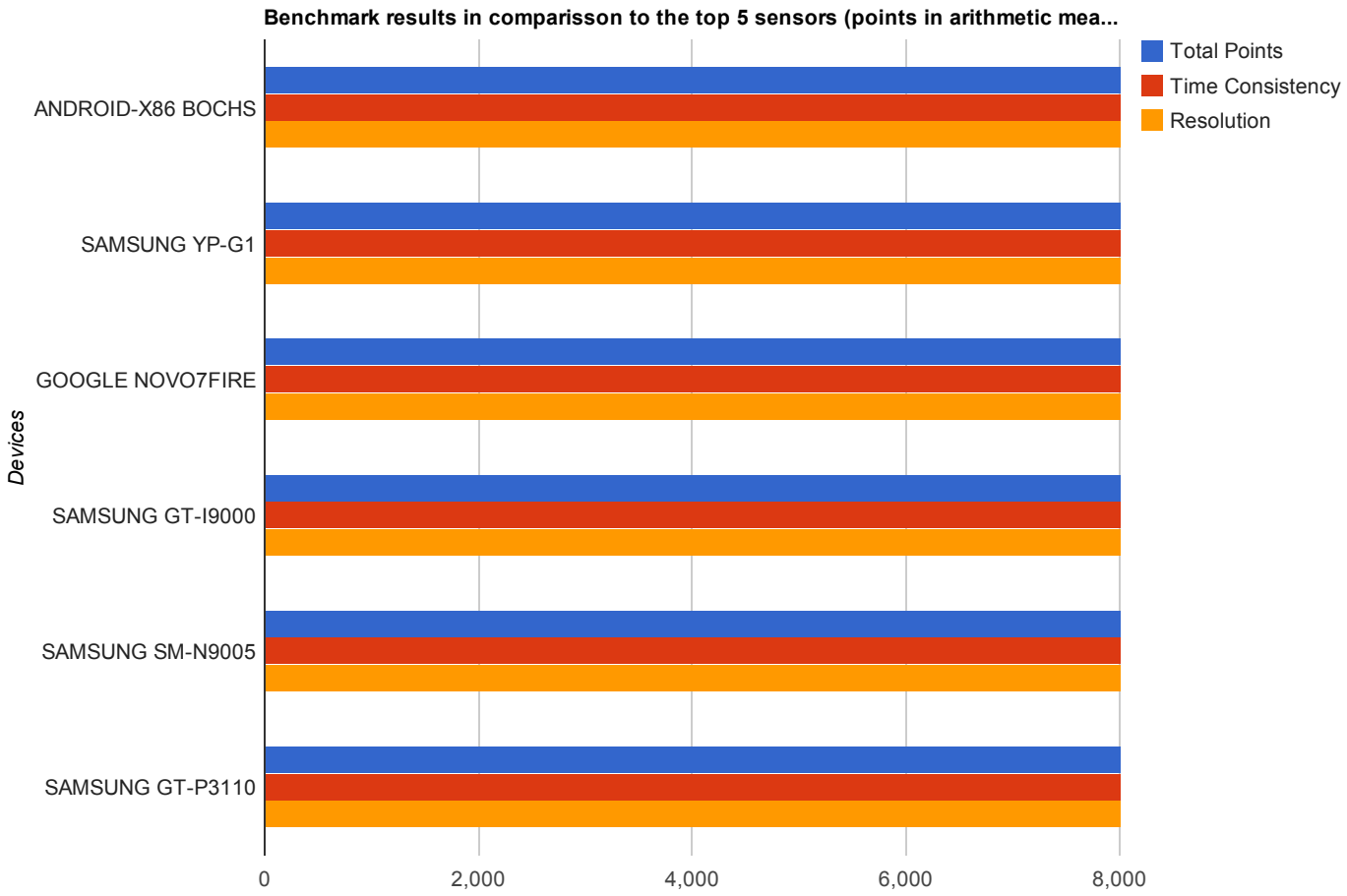
Device

Brand	Google
Manufacturer	Samsung
Device	crespo
Model	Nexus S
Product	soju
Hardware	herring
Board	herring

Sensor Benchmarks

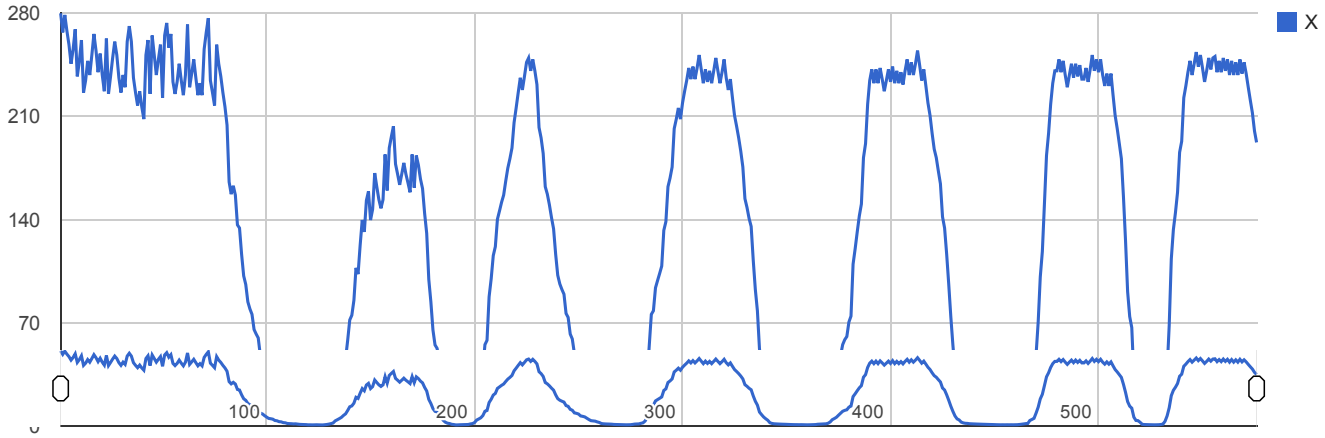
Light: Light sensor Emulation!!!! by Columbia University - NYC - Thesis - Raghavan Santhanam

Benchmark



Points

Total	5224
Events	576
Deviation Timepitch	1913
Resolution	2711
Differing Values	600



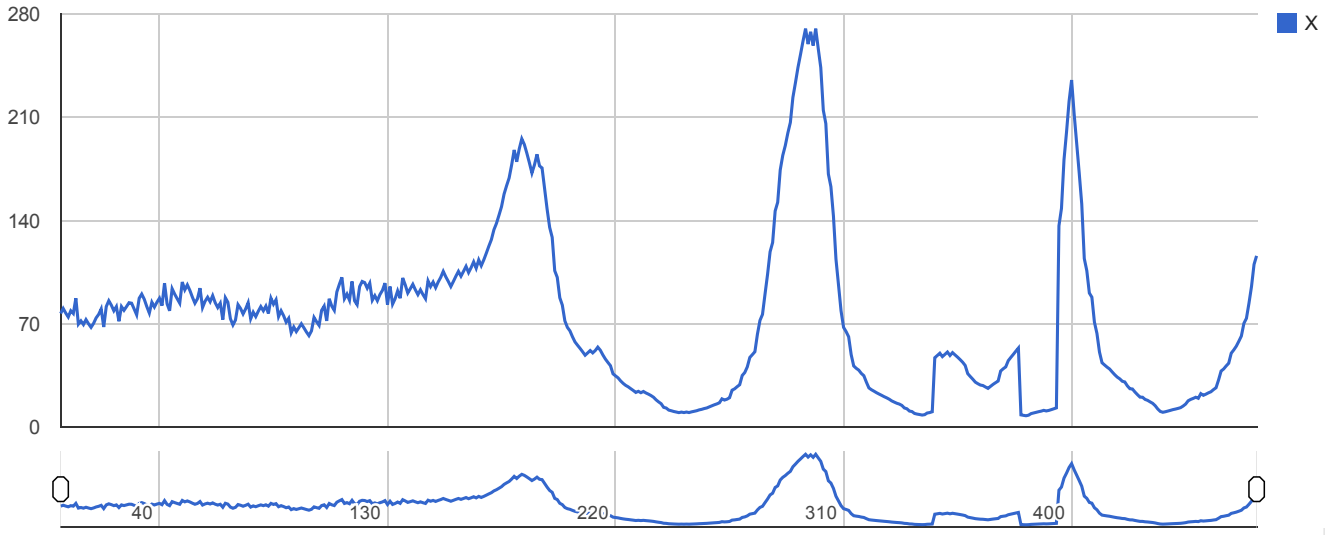
Download all values as JSON array

Number of events	576
Timepitch (minimum)	2121762 µs
Timepitch (arithmetic mean)	16320909.4747826 µs
Timepitch (standard deviation)	27553568.4723946 µs
Minimum Resolution X	0.021361351 lux
Minimum X	4.397545 lux
Mean X	132.886638003919 lux
Maximum X	279.8769 lux
Standard Deviation X	99.2027717219355 lux
Android Version	4.3.1
App Version	1.0
App Version Code	9
Battery	100%
Charging	0

SensMark results based on all executed initializations and benchmarks with this specific sensor

Minimum Timepitch	1055755 µs
Minimum Resolution X	0.021361351 lux
Minimum X	4.397545 lux
Maximum X	279.8769 lux

Initialization



Download all values as JSON array

Number of events	473
Timepitch (minimum)	1055755 µs
Timepitch (arithmetic mean)	19973191.5699153 µs
Timepitch (standard deviation)	73148479.5193068 µs
Minimum Resolution X	0.07984924 lux
Minimum X	7.642827 lux
Maximum X	270.10687 lux

About Device

Technical sensor specifications by device

Maximum Range	0 lux	
Minimum Delay	0 µs	The minimum delay allowed between two events in microsecond or zero if this sensor only returns a value when the data it's measuring changes.
Power	0.75 mA	The power in mA used by this sensor while in use.
Resolution	1 lux	
Version	1	

Note: In our research we found out that the specifications above do not have to be true.

Device

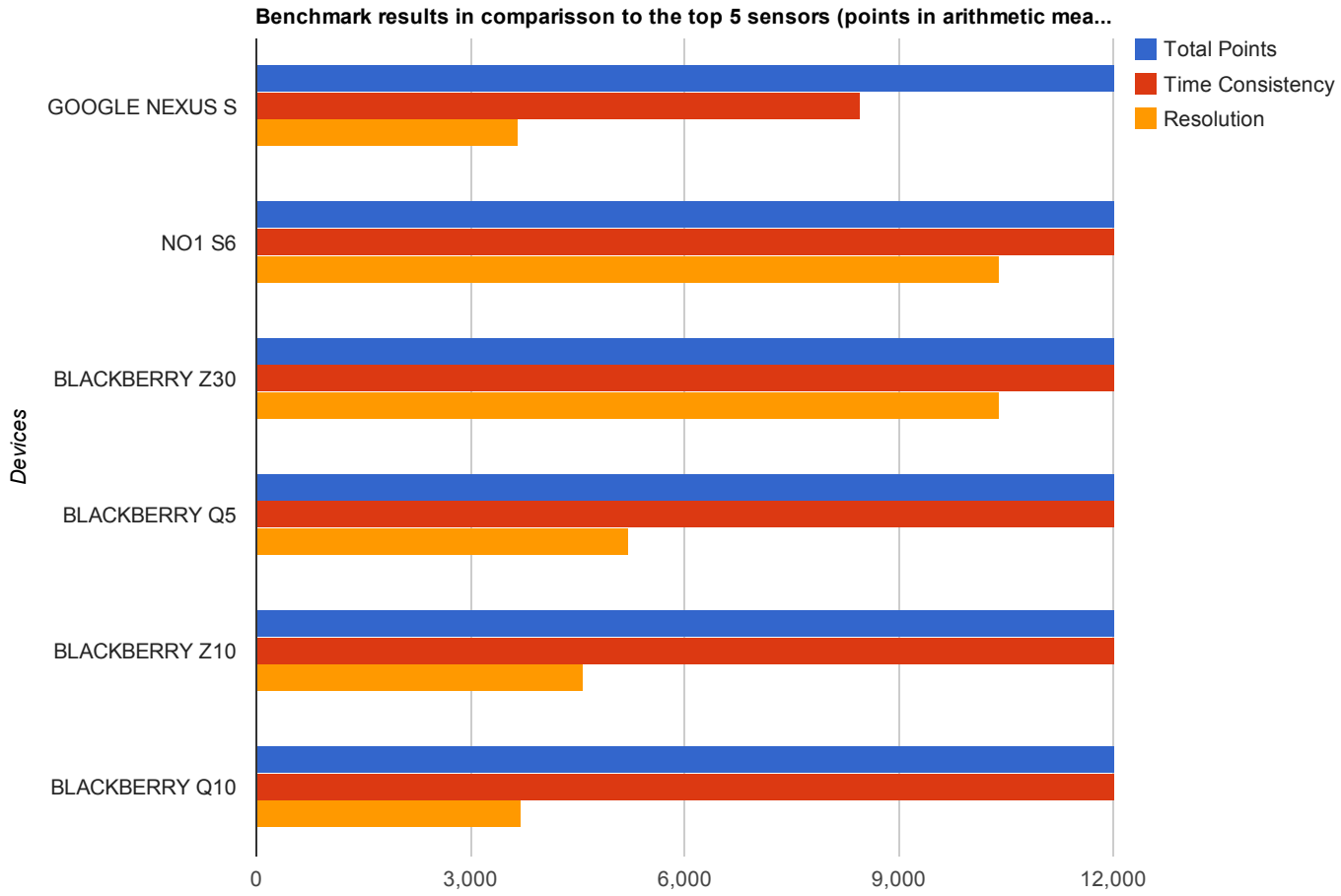
Brand	Android-x86
Manufacturer	Bochs
Device	x86
Model	Bochs
Product	android_x86
Hardware	android_x86
Board	unknown

↑

Sensor Benchmarks

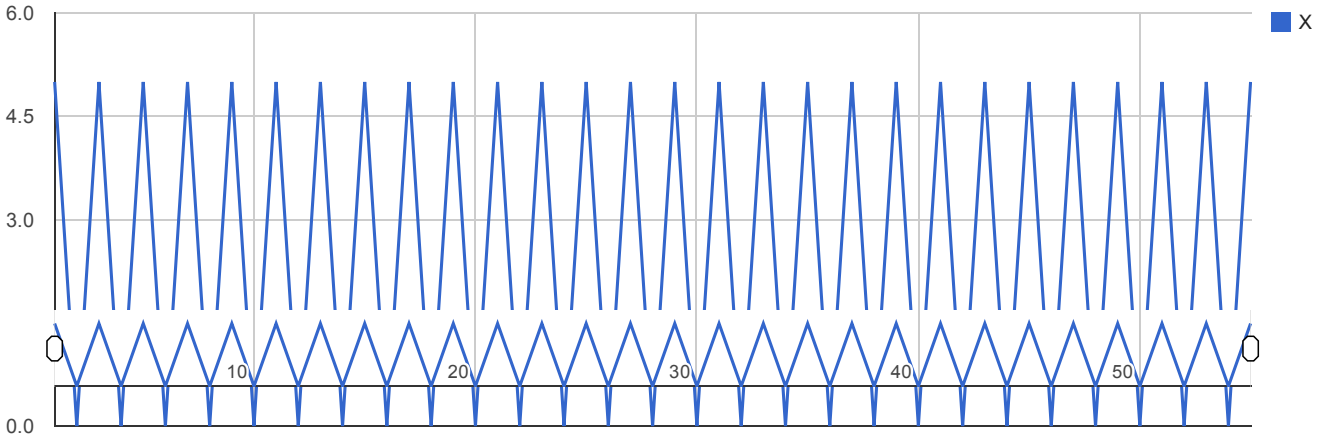
Proximity: GP2A Proximity sensor by Sharp

Benchmark



Points

Total	1214
Events	55
Deviation Timepitch	845
Resolution	367
Differing Values	2

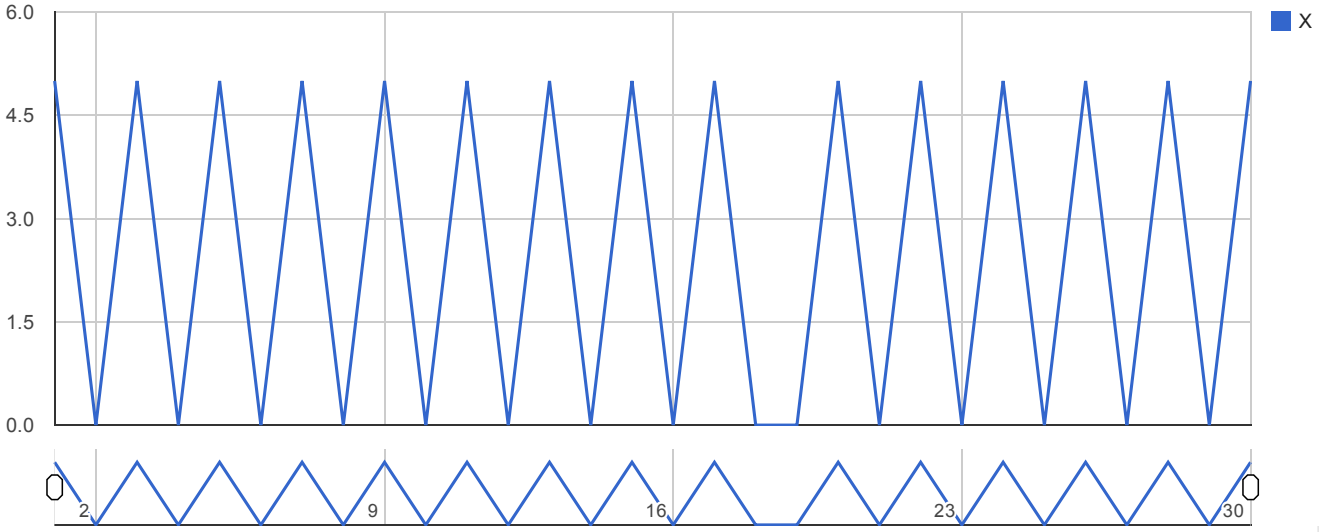


Number of events	55
Timepitch (minimum)	14869000 µs
Timepitch (arithmetic mean)	181887724.481481 µs
Timepitch (standard deviation)	429391143.993835 µs
Minimum Resolution X	5 cm
Minimum X	0 cm
Mean X	2.54545454545455 cm
Maximum X	5 cm
Standard Deviation X	2.52262489554756 cm
Android Version	4.3
App Version	1.0
App Version Code	9
Battery	39%
Charging	1

SensMark results based on all executed initializations and benchmarks with this specific sensor

Minimum Timepitch	14869000 µs
Minimum Resolution X	5 cm
Minimum X	0 cm
Maximum X	5 cm

Initialization



Download all values as JSON array

Number of events	30
Timepitch (minimum)	52048000 µs
Timepitch (arithmetic mean)	288402337.931034 µs
Timepitch (standard deviation)	517442017.777228 µs
Minimum Resolution X	5 cm
Minimum X	0 cm
Maximum X	5 cm

About Device

Technical sensor specifications by device

Maximum Range	5 cm	
Minimum Delay	0 µs	The minimum delay allowed between two events in microsecond or zero if this sensor only returns a value when the data it's measuring changes.
Power	0.75 mA	The power in mA used by this sensor while in use.
Resolution	1 cm	
Version	1	

Note: In our research we found out that the specifications above do not have to be true.

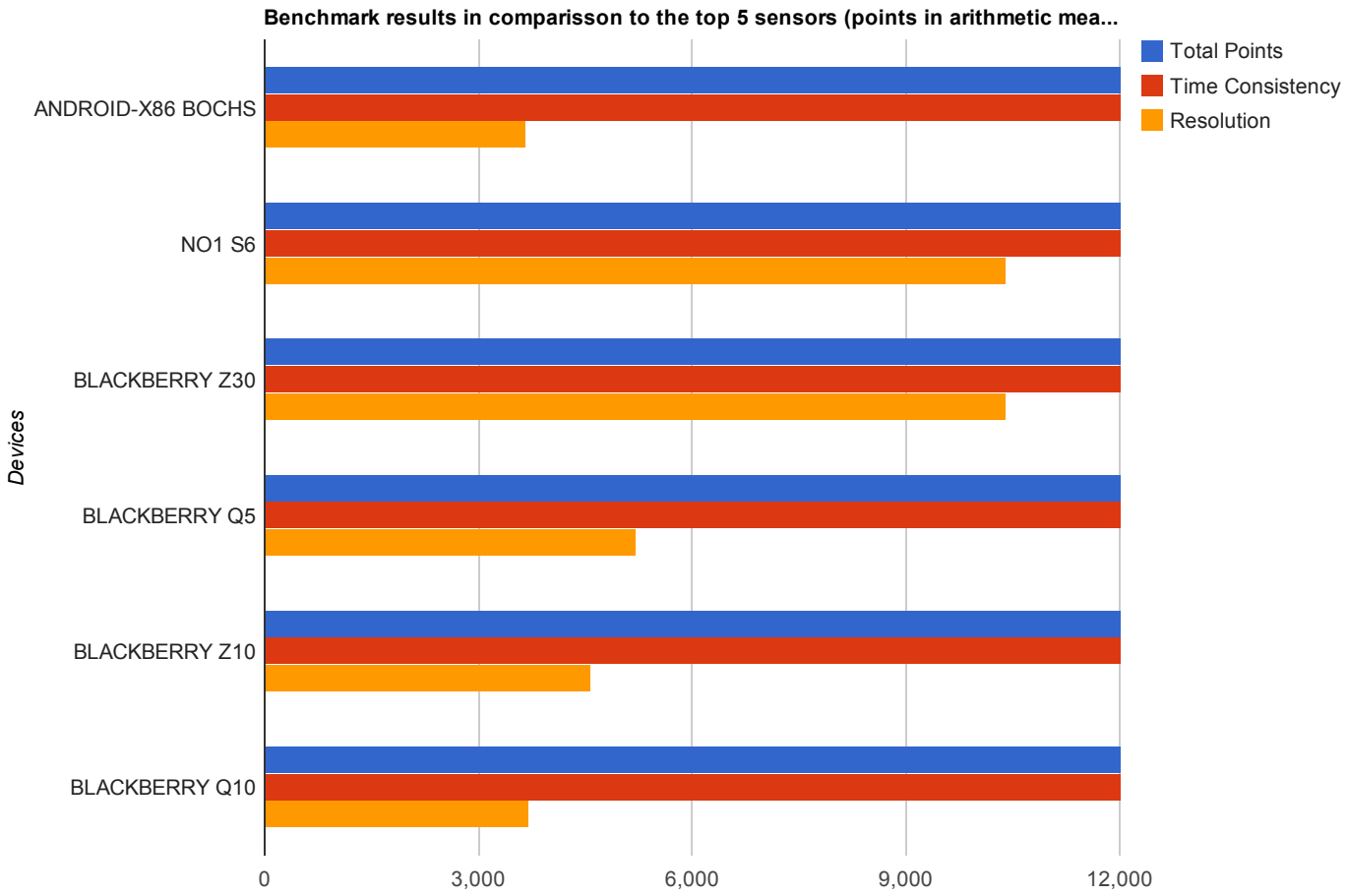
Device

Brand	Google
Manufacturer	Samsung
Device	crespo
Model	Nexus S
Product	soju
Hardware	herring
Board	herring

Sensor Benchmarks

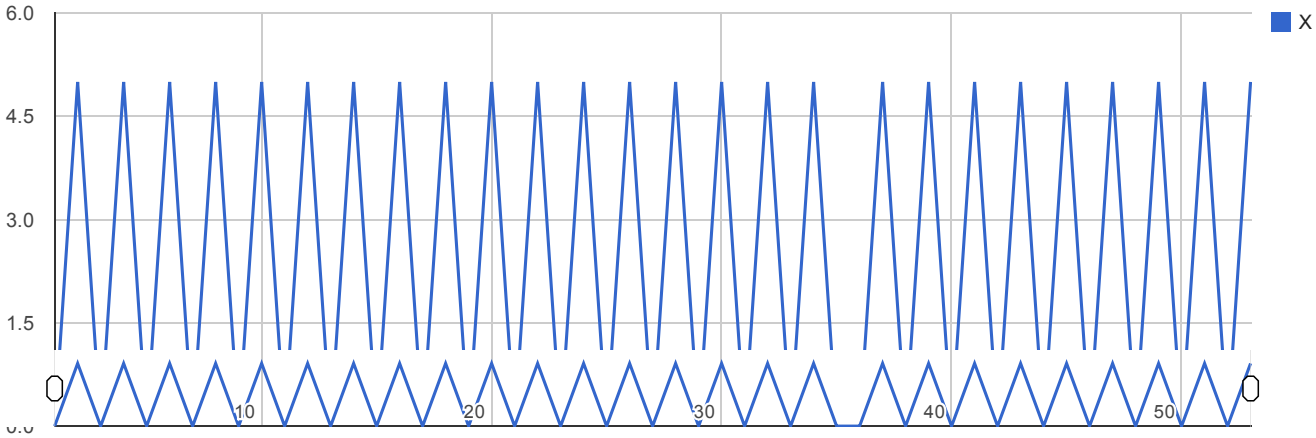
Proximity: Proximity sensor Emulation!!!! by Columbia University - NYC - Thesis - Raghavan Santhanam

Benchmark



Points

Total	1647
Events	53
Deviation Timepitch	1278
Resolution	367
Differing Values	2



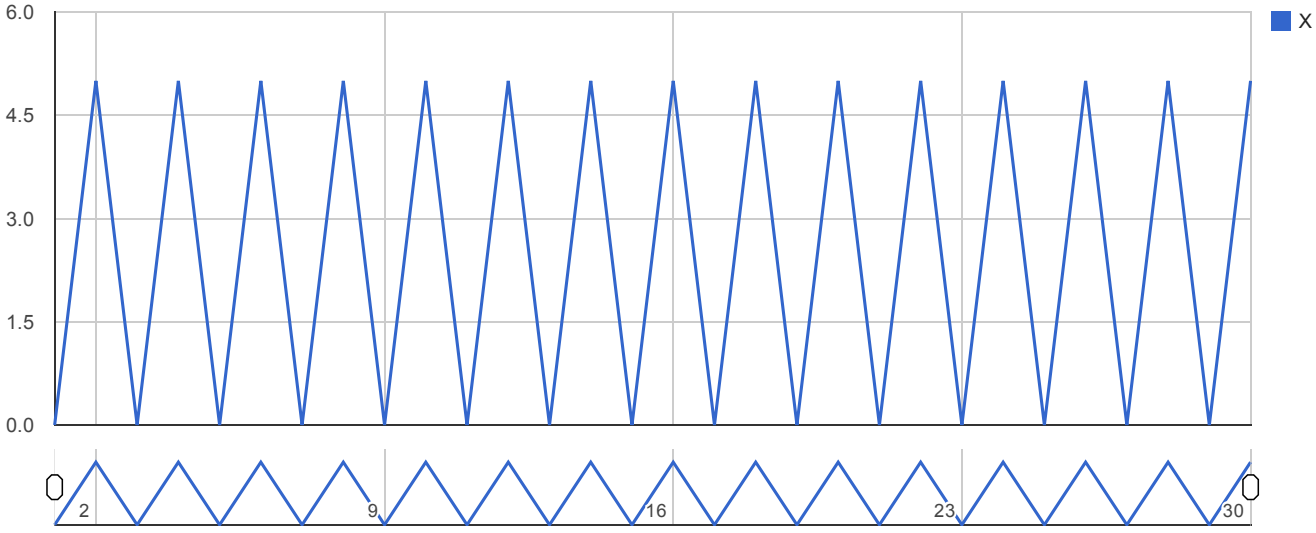
Download all values as JSON array

Number of events	53
Timepitch (minimum)	3304838 µs
Timepitch (arithmetic mean)	130092980.519231 µs
Timepitch (standard deviation)	161799650.769043 µs
Minimum Resolution X	5 cm
Minimum X	0 cm
Mean X	2.45283018867924 cm
Maximum X	5 cm
Standard Deviation X	2.5234746934142 cm
Android Version	4.3.1
App Version	1.0
App Version Code	9
Battery	100%
Charging	0

SensMark results based on all executed initializations and benchmarks with this specific sensor

Minimum Timepitch	3265474 µs
Minimum Resolution X	5 cm
Minimum X	0 cm
Maximum X	5 cm

Initialization



Download all values as JSON array

Number of events	30
Timepitch (minimum)	9164813 µs
Timepitch (arithmetic mean)	197116601.275862 µs
Timepitch (standard deviation)	178443427.454734 µs
Minimum Resolution X	5 cm
Minimum X	0 cm
Maximum X	5 cm

About Device

Technical sensor specifications by device

Maximum Range	5 cm	
Minimum Delay	0 µs	The minimum delay allowed between two events in microsecond or zero if this sensor only returns a value when the data it's measuring changes.
Power	0.75 mA	The power in mA used by this sensor while in use.
Resolution	1 cm	
Version	1	

Note: In our research we found out that the specifications above do not have to be true.

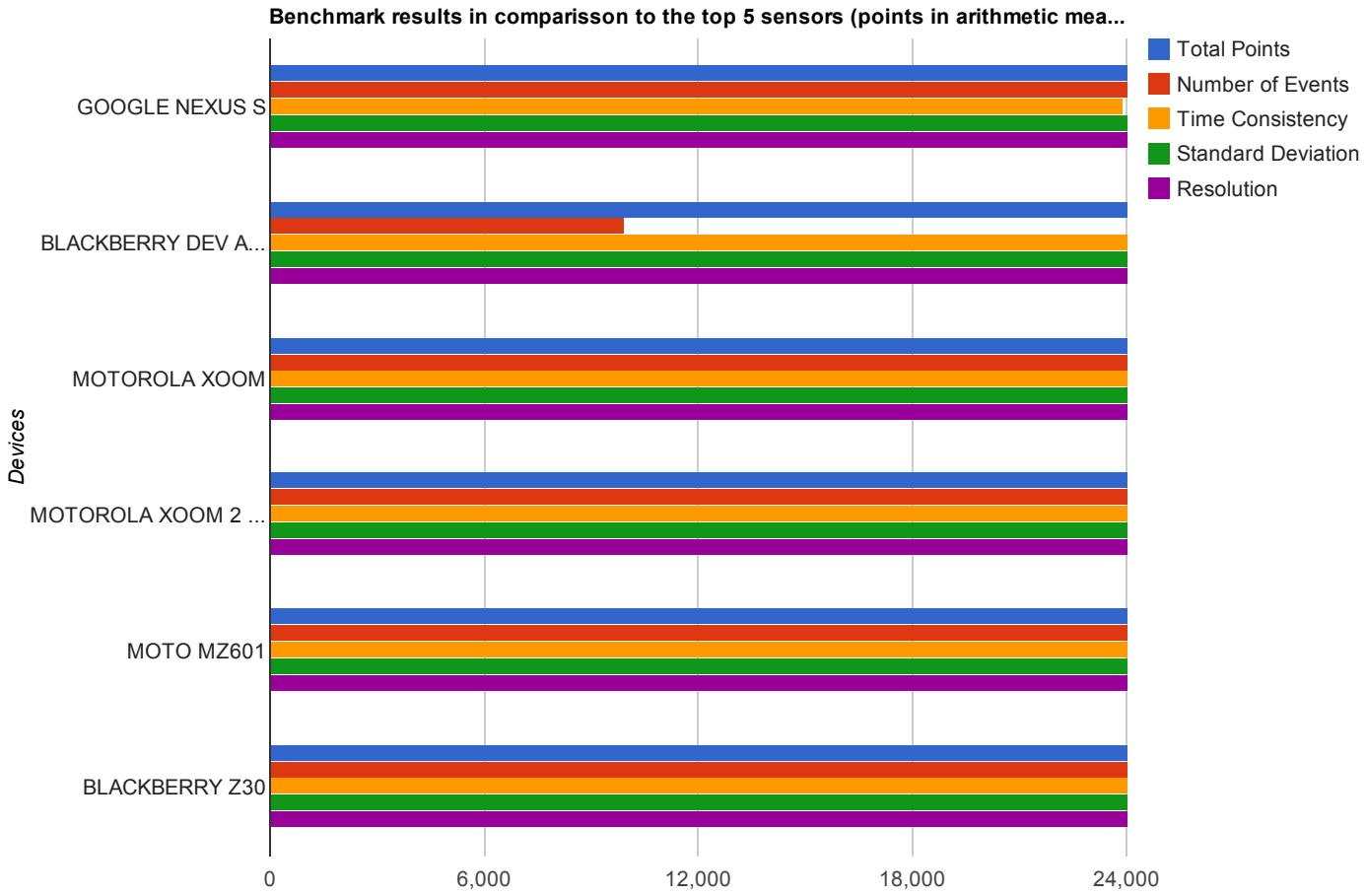
Device

Brand	Android-x86
Manufacturer	Bochs
Device	x86
Model	Bochs
Product	android_x86
Hardware	android_x86
Board	unknown

Sensor Benchmarks

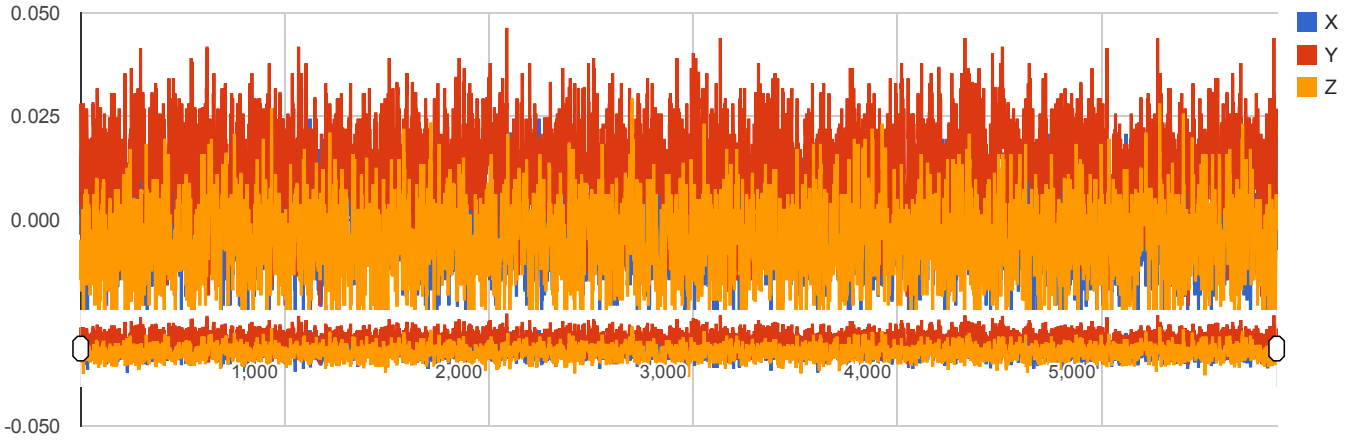
Gyroscope: K3G Gyroscope sensor by STMicroelectronics

Benchmark



Points

Total	16697
Events	5853
Deviation Timepitch	2390
Deviation Values	4023
Resolution	4431

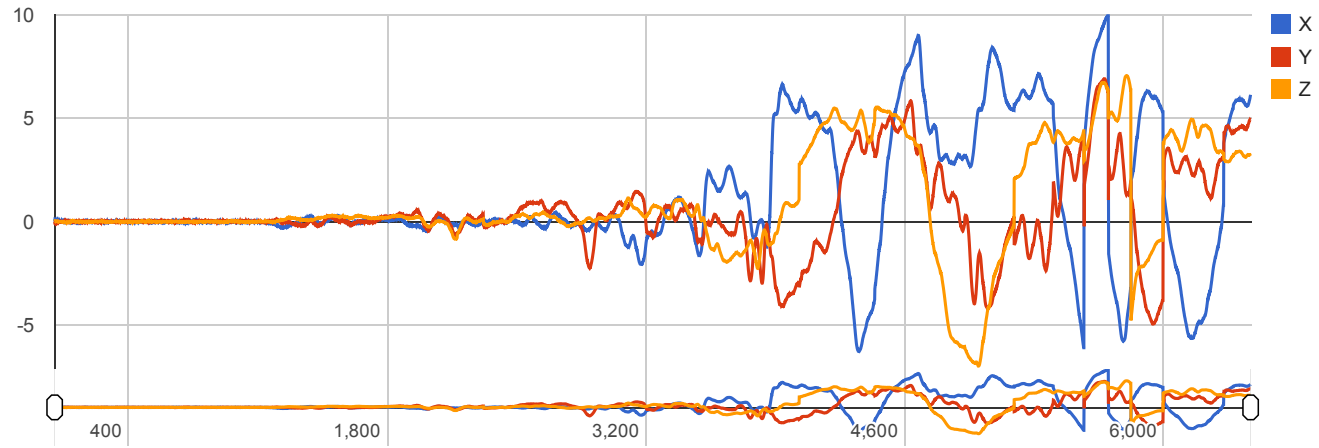


Number of events	5853
Timepitch (minimum)	5000 µs
Timepitch (arithmetic mean)	1323039.81544771 µs
Timepitch (standard deviation)	10146717.8264915 µs
Minimum Resolution X	0.0012217294 rad/s
Min. Resolution Y	0.0012217294 rad/s
Min. Resolution Z	0.0012217294 rad/s
Minimum X	-0.03176499 rad/s
Mean X	-0.00288472839172242 rad/s
Maximum X	0.02443461 rad/s
Minimum Y	-0.021991149 rad/s
Mean Y	0.0121271310204639 rad/s
Maximum Y	0.04642576 rad/s
Minimum Z	-0.037873644 rad/s
Mean Z	-0.00493075645573842 rad/s
Maximum Z	0.029321533 rad/s
Standard Deviation X	0.00873989164838175 rad/s
Standard Deviation Y	0.0102265468457807 rad/s
Standard Deviation Z	0.00931612656677903 rad/s
Android Version	4.3
App Version	1.0
App Version Code	9
Battery	72%
Charging	0

SensMark results based on all executed initializations and benchmarks with this specific sensor

Minimum Timepitch	1000 µs
Minimum Resolution X	0.0012216568 rad/s
Min. Resolution Y	0.0012216568 rad/s
Min. Resolution Z	0.0012216568 rad/s
Minimum X	-6.2540383 rad/s
Maximum X	9.968099 rad/s
Minimum Y	-4.9565606 rad/s
Maximum Y	6.917438 rad/s
Minimum Z	-6.994407 rad/s
Maximum Z	7.064046 rad/s

Initialization



Download all values as JSON array

Number of events	6472
Timepitch (minimum)	1000 µs
Timepitch (arithmetic mean)	1649928.28774532 µs
Timepitch (standard deviation)	19671700.841507 µs
Minimum Resolution X	0.0012216568 rad/s
Min. Resolution Y	0.0012216568 rad/s
Min. Resolution Z	0.0012216568 rad/s
Minimum X	-6.2540383 rad/s
Maximum X	9.968099 rad/s
Minimum Y	-4.9565606 rad/s
Maximum Y	6.917438 rad/s
Minimum Z	-6.994407 rad/s
Maximum Z	7.064046 rad/s

About Device

Technical sensor specifications by device

Maximum Range	34.906586 rad/s	
Minimum Delay	1190 µs	The minimum delay allowed between two events in microsecond or zero if this sensor only returns a value when the data it's measuring changes.
Power	6.1 mA	The power in mA used by this sensor while in use.
Resolution	1 rad/s	
Version	1	

Note: In our research we found out that the specifications above do not have to be true.

Device

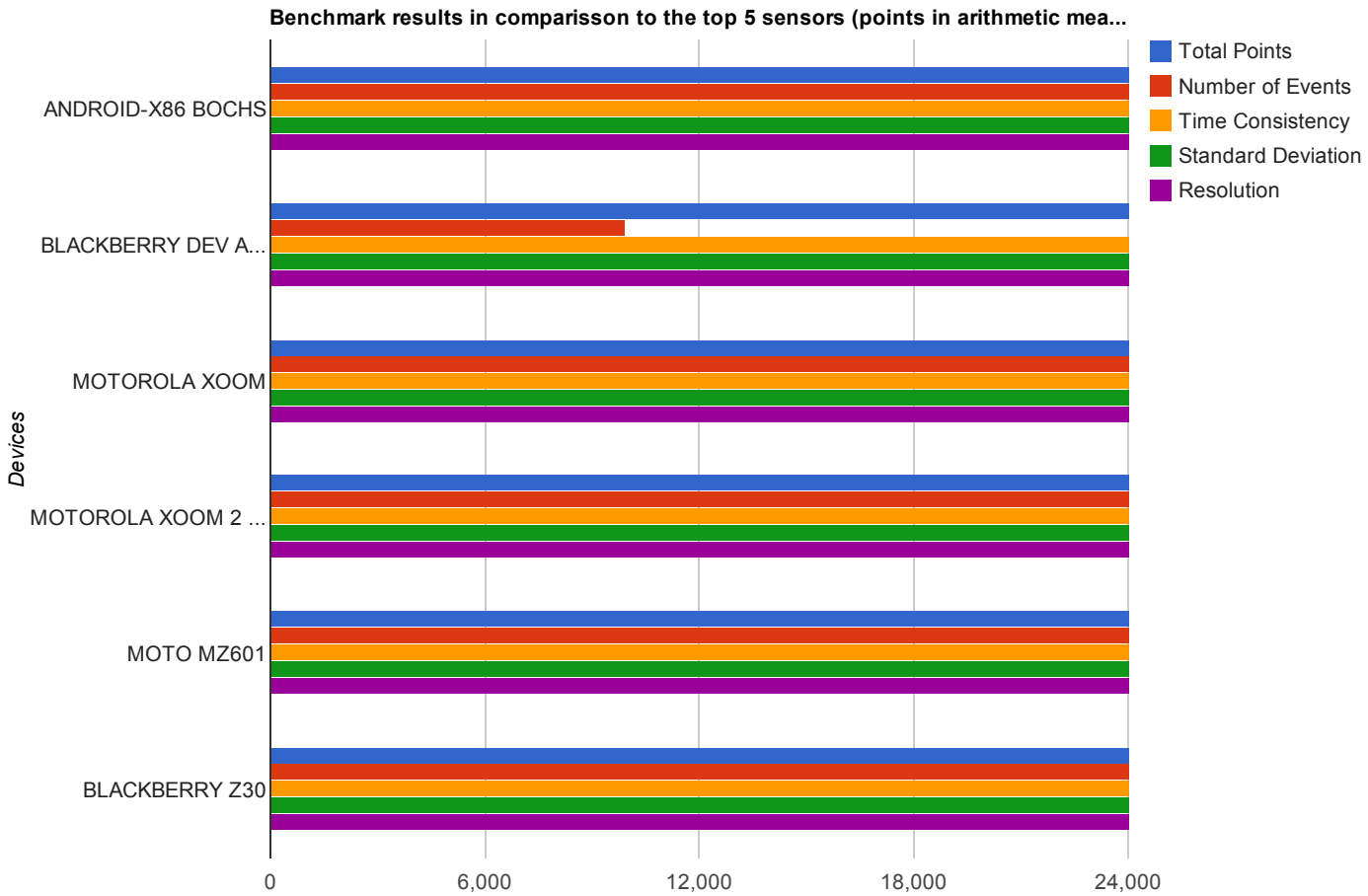
Brand	Google
Manufacturer	Samsung
Device	crespo
Model	Nexus S
Product	soju
Hardware	herring
Board	herring

↑

Sensor Benchmarks

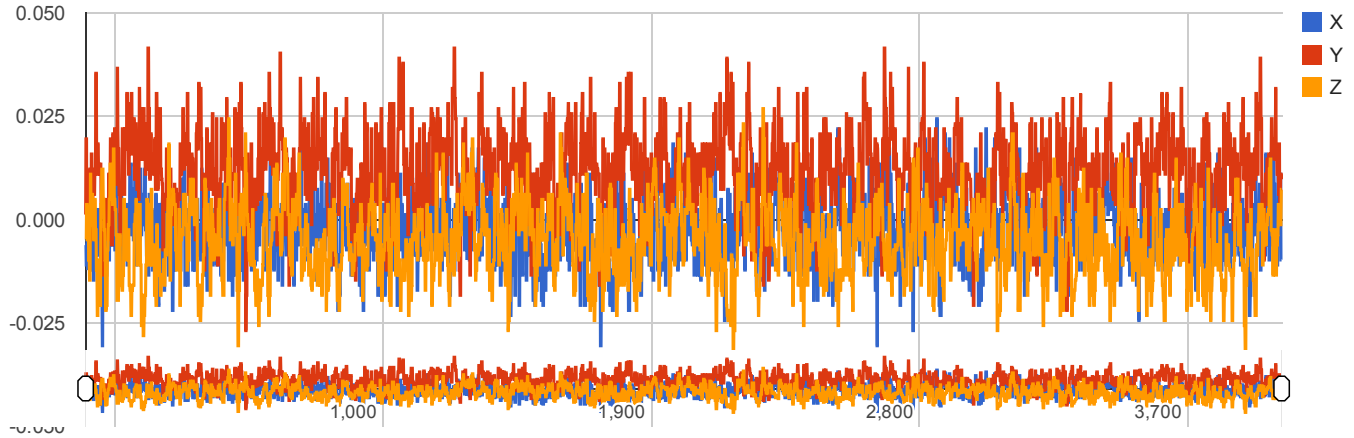
Gyroscope: Gyroscope sensor Emulation!!!! by Columbia University - NYC - Thesis - Raghavan Santhanam

Benchmark



Points

Total	15875
Events	4013
Deviation Timepitch	3416
Deviation Values	4015
Resolution	4431



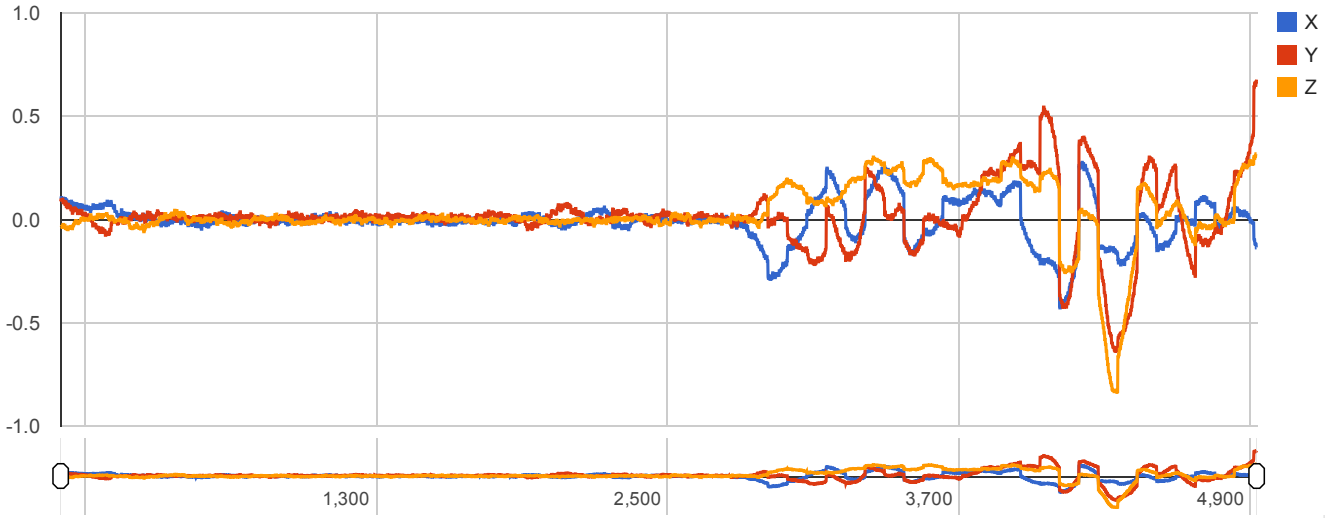
Download all values as JSON array

Number of events	4013
Timepitch (minimum)	1030316 µs
Timepitch (arithmetic mean)	2485833.26495513 µs
Timepitch (standard deviation)	4688438.78654003 µs
Minimum Resolution X	0.0012217294 rad/s
Min. Resolution Y	0.0012217294 rad/s
Min. Resolution Z	0.0012217294 rad/s
Minimum X	-0.030543262 rad/s
Mean X	-0.00238135454245259 rad/s
Maximum X	0.02443461 rad/s
Minimum Y	-0.02687807 rad/s
Mean Y	0.0117317179953675 rad/s
Maximum Y	0.04153884 rad/s
Minimum Z	-0.03176499 rad/s
Mean Z	-0.00502757469160586 rad/s
Maximum Z	0.02687807 rad/s
Standard Deviation X	0.00884474442747728 rad/s
Standard Deviation Y	0.0103738133671091 rad/s
Standard Deviation Z	0.00956108287150725 rad/s
Android Version	4.3.1
App Version	1.0
App Version Code	9
Battery	100%
Charging	0

SensMark results based on all executed initializations and benchmarks with this specific sensor

Minimum Timepitch	1030316 µs
Minimum Resolution X	0.0012217164 rad/s
Min. Resolution Y	0.0012217164 rad/s
Min. Resolution Z	0.0012217164 rad/s
Minimum X	-0.42760566 rad/s
Maximum X	0.2773328 rad/s
Minimum Y	-0.6377433 rad/s
Maximum Y	0.6780604 rad/s
Minimum Z	-0.8368854 rad/s
Maximum Z	0.31887165 rad/s

Initialization



Download all values as JSON array

Number of events	4924
Timepitch (minimum)	1044881 µs
Timepitch (arithmetic mean)	2031962.296364 µs
Timepitch (standard deviation)	580817.836400734 µs
Minimum Resolution X	0.0012217164 rad/s
Min. Resolution Y	0.0012217164 rad/s
Min. Resolution Z	0.0012217164 rad/s
Minimum X	-0.42760566 rad/s
Maximum X	0.2773328 rad/s
Minimum Y	-0.6377433 rad/s
Maximum Y	0.6780604 rad/s
Minimum Z	-0.8368854 rad/s
Maximum Z	0.31887165 rad/s

About Device

Technical sensor specifications by device

Maximum Range	34.906586 rad/s	
Minimum Delay	1190 µs	The minimum delay allowed between two events in microsecond or zero if this sensor only returns a value when the data it's measuring changes.
Power	6.1 mA	The power in mA used by this sensor while in use.
Resolution	1 rad/s	
Version	1	

Note: In our research we found out that the specifications above do not have to be true.

Device

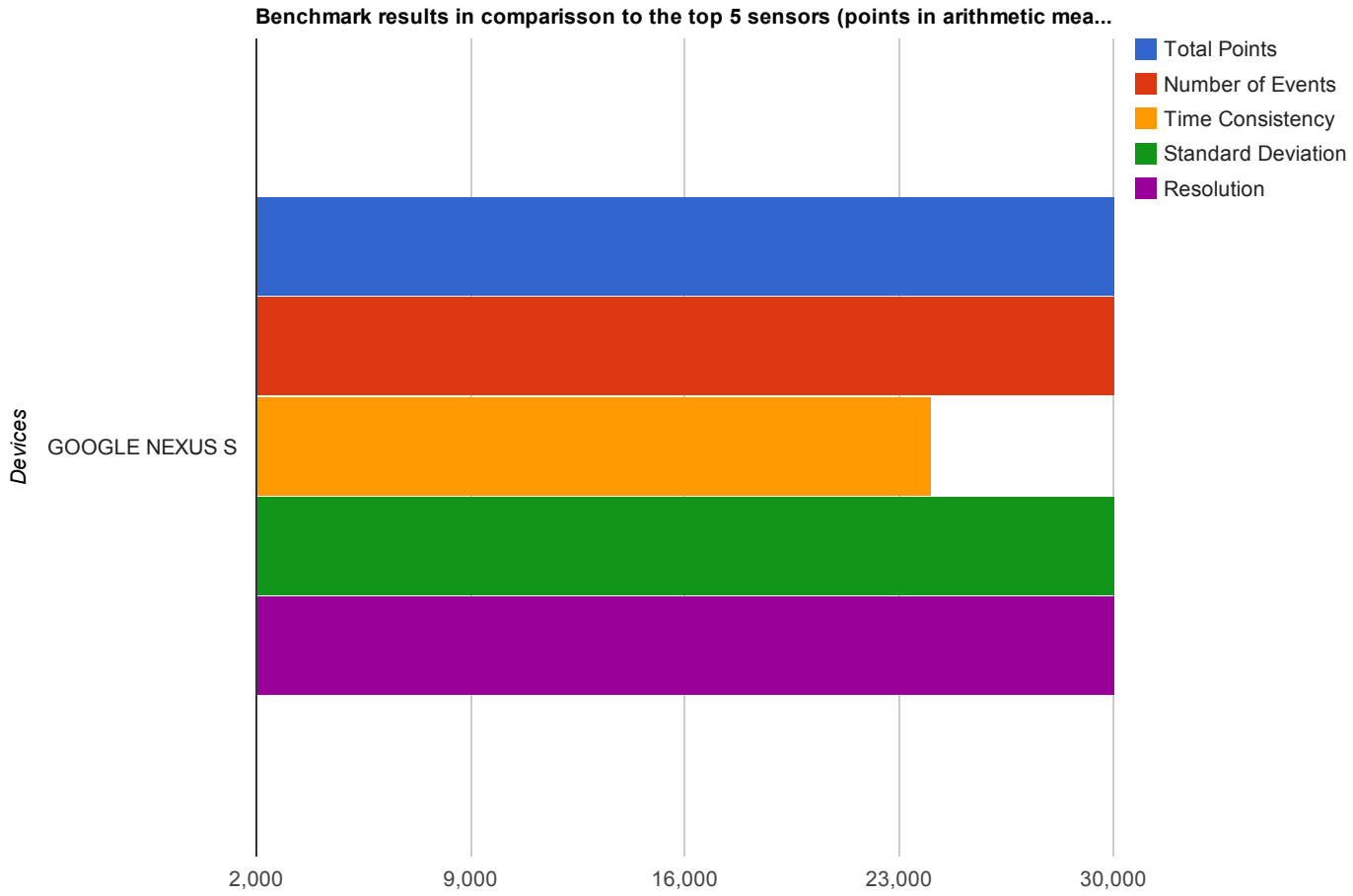
Brand	Android-x86
Manufacturer	Bochs
Device	x86
Model	Bochs
Product	android_x86
Hardware	android_x86
Board	unknown

↑

Sensor Benchmarks

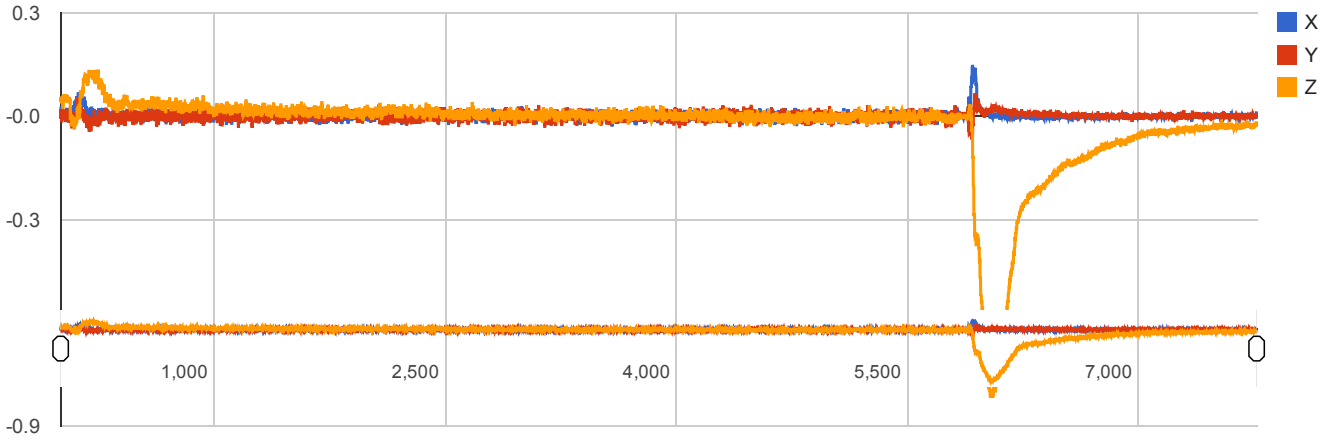
Gyroscope: Corrected Gyroscope Sensor by Google Inc.

Benchmark



Points

Total	23700
Events	7764
Deviation Timepitch	2408
Deviation Values	3778
Resolution	9750

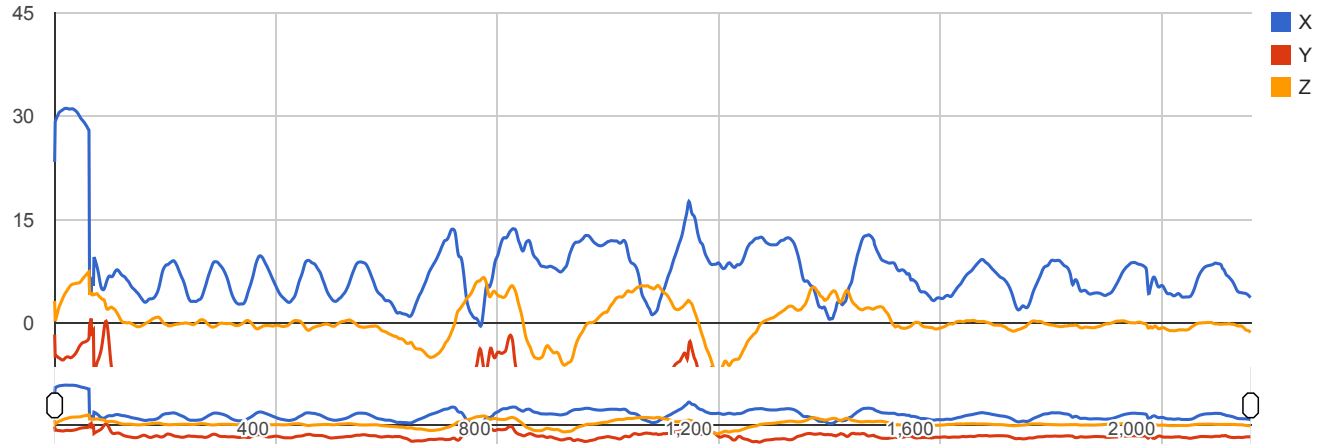


Number of events	7764
Timepitch (minimum)	5000 µs
Timepitch (arithmetic mean)	2712234.04804844 µs
Timepitch (standard deviation)	46591804.9106864 µs
Minimum Resolution X	4.656613e-09 rad/s
Min. Resolution Y	3.4458935e-08 rad/s
Min. Resolution Z	3.7252903e-09 rad/s
Minimum X	-0.029162565 rad/s
Mean X	0.00135079758898791 rad/s
Maximum X	0.14878188 rad/s
Minimum Y	-0.06704378 rad/s
Mean Y	0.00105244149020005 rad/s
Maximum Y	0.06387907 rad/s
Minimum Z	-0.81840205 rad/s
Mean Z	-0.0306139868136419 rad/s
Maximum Z	0.13552837 rad/s
Standard Deviation X	0.0124841781918444 rad/s
Standard Deviation Y	0.0106674534661779 rad/s
Standard Deviation Z	0.124873303874371 rad/s
Android Version	4.3
App Version	1.0
App Version Code	9
Battery	72%
Charging	0

SensMark results based on all executed initializations and benchmarks with this specific sensor

Minimum Timepitch	5000 µs
Minimum Resolution X	4.656613e-09 rad/s
Min. Resolution Y	3.4458935e-08 rad/s
Min. Resolution Z	3.7252903e-09 rad/s
Minimum X	-0.47589684 rad/s
Maximum X	31.16835 rad/s
Minimum Y	-14.308565 rad/s
Maximum Y	0.62288666 rad/s
Minimum Z	-7.6624045 rad/s
Maximum Z	7.415291 rad/s

Initialization



Download all values as JSON array

Number of events	2160
Timepitch (minimum)	2019000 µs
Timepitch (arithmetic mean)	4798122.68133396 µs
Timepitch (standard deviation)	7459310.05262057 µs
Minimum Resolution X	1.4781952e-05 rad/s
Min. Resolution Y	3.1471252e-05 rad/s
Min. Resolution Z	1.013279e-06 rad/s
Minimum X	-0.47589684 rad/s
Maximum X	31.16835 rad/s
Minimum Y	-14.308565 rad/s
Maximum Y	0.62288666 rad/s
Minimum Z	-7.6624045 rad/s
Maximum Z	7.415291 rad/s

About Device

Technical sensor specifications by device

Maximum Range	34.906586 rad/s	
Minimum Delay	1190 µs	The minimum delay allowed between two events in microsecond or zero if this sensor only returns a value when the data it's measuring changes.
Power	13.13 mA	The power in mA used by this sensor while in use.
Resolution	1 rad/s	
Version	1	

Note: In our research we found out that the specifications above do not have to be true.

Device

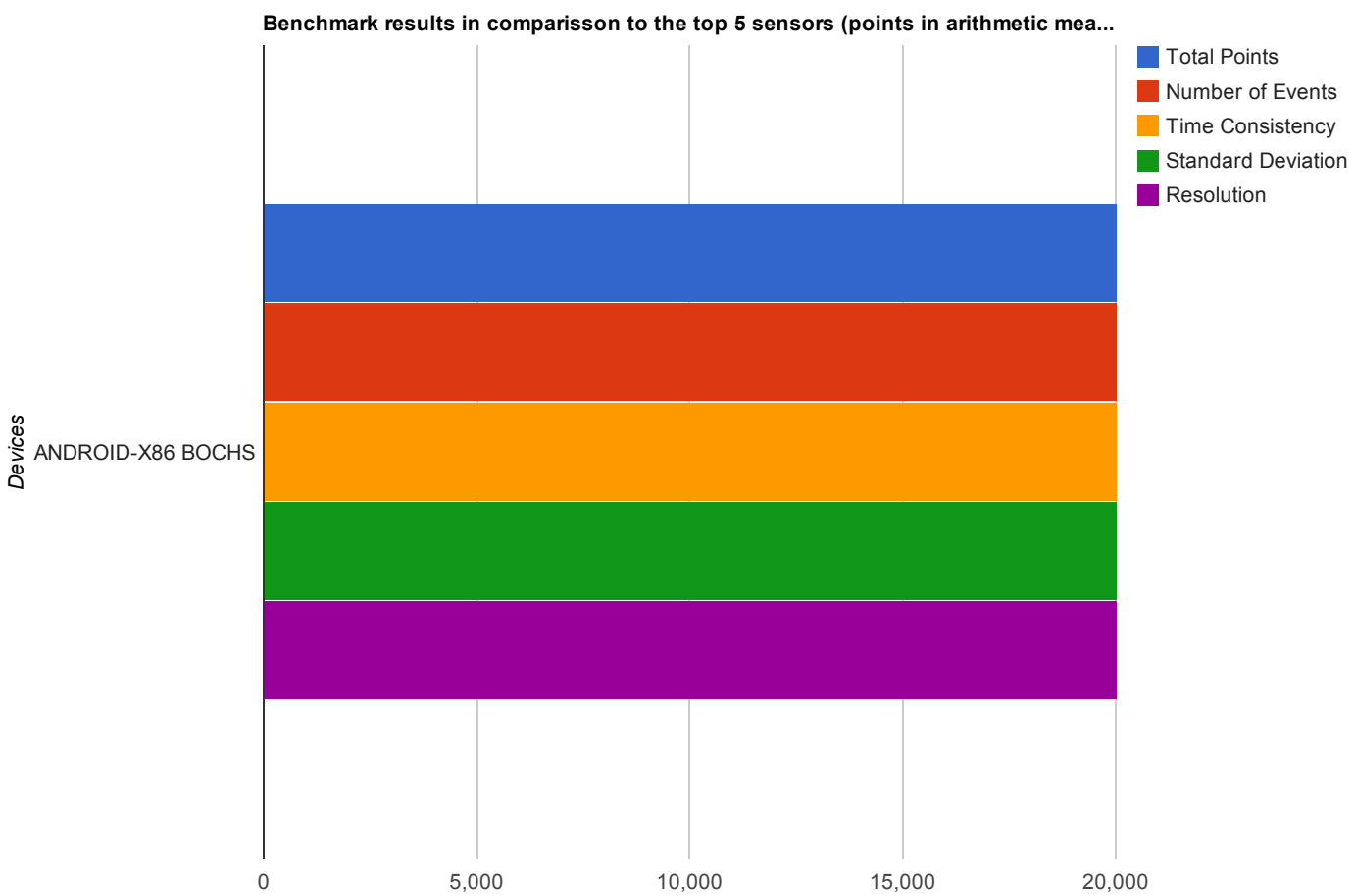
Brand	Google
Manufacturer	Samsung
Device	crespo
Model	Nexus S
Product	soju
Hardware	herring
Board	herring

↑

Sensor Benchmarks

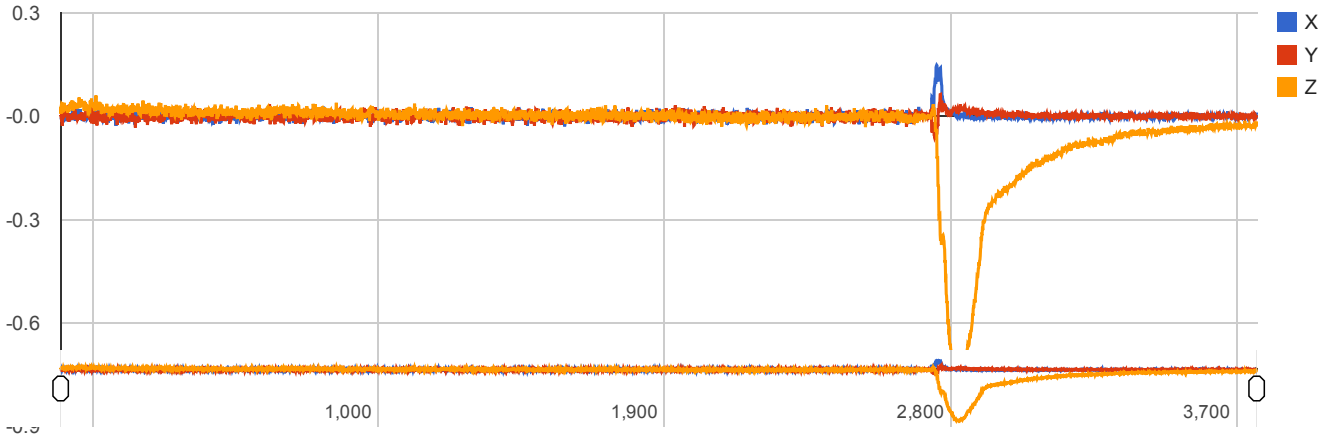
Gyroscope: Corrected Gyroscope Sensor Emulation!!!! by Google Inc.

Benchmark



Points

Total	18521
Events	3762
Deviation Timepitch	2311
Deviation Values	3786
Resolution	8662



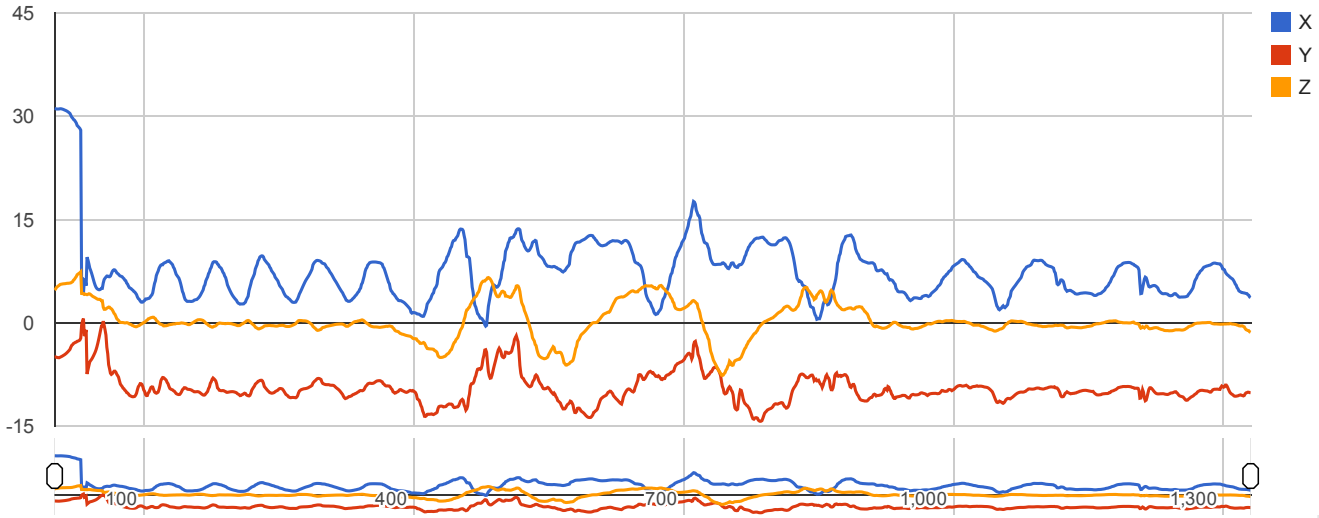
Download all values as JSON array

Number of events	3762
Timepitch (minimum)	1029845 µs
Timepitch (arithmetic mean)	5015575.6833289 µs
Timepitch (standard deviation)	14902575.0686187 µs
Minimum Resolution X	7.8231096e-08 rad/s
Min. Resolution Y	2.4598558e-07 rad/s
Min. Resolution Z	4.004687e-08 rad/s
Minimum X	-0.029162565 rad/s
Mean X	0.000703288323891424 rad/s
Maximum X	0.14878188 rad/s
Minimum Y	-0.06704378 rad/s
Mean Y	0.00102522614198671 rad/s
Maximum Y	0.06387907 rad/s
Minimum Z	-0.81840205 rad/s
Mean Z	-0.0373121408507692 rad/s
Maximum Z	0.060750667 rad/s
Standard Deviation X	0.0125816132428213 rad/s
Standard Deviation Y	0.0102227097479402 rad/s
Standard Deviation Z	0.125127977494517 rad/s
Android Version	4.3.1
App Version	1.0
App Version Code	9
Battery	100%
Charging	0

SensMark results based on all executed initializations and benchmarks with this specific sensor

Minimum Timepitch	1029845 µs
Minimum Resolution X	7.8231096e-08 rad/s
Min. Resolution Y	2.4598558e-07 rad/s
Min. Resolution Z	4.004687e-08 rad/s
Minimum X	-0.47589684 rad/s
Maximum X	31.107265 rad/s
Minimum Y	-14.308565 rad/s
Maximum Y	0.62288666 rad/s
Minimum Z	-7.6514087 rad/s
Maximum Z	7.415291 rad/s

Initialization



Download all values as JSON array

Number of events	1330
Timepitch (minimum)	1034347 µs
Timepitch (arithmetic mean)	7456090.7945824 µs
Timepitch (standard deviation)	17379266.2977709 µs
Minimum Resolution X	2.0980835e-05 rad/s
Min. Resolution Y	4.1007996e-05 rad/s
Min. Resolution Z	1.013279e-06 rad/s
Minimum X	-0.47589684 rad/s
Maximum X	31.141472 rad/s
Minimum Y	-14.308565 rad/s
Maximum Y	0.62288666 rad/s
Minimum Z	-7.6514087 rad/s
Maximum Z	7.415291 rad/s

About Device

Technical sensor specifications by device

Maximum Range	34.906586 rad/s	
Minimum Delay	1190 µs	The minimum delay allowed between two events in microsecond or zero if this sensor only returns a value when the data it's measuring changes.
Power	13.13 mA	The power in mA used by this sensor while in use.
Resolution	1 rad/s	
Version	1	

Note: In our research we found out that the specifications above do not have to be true.

Device

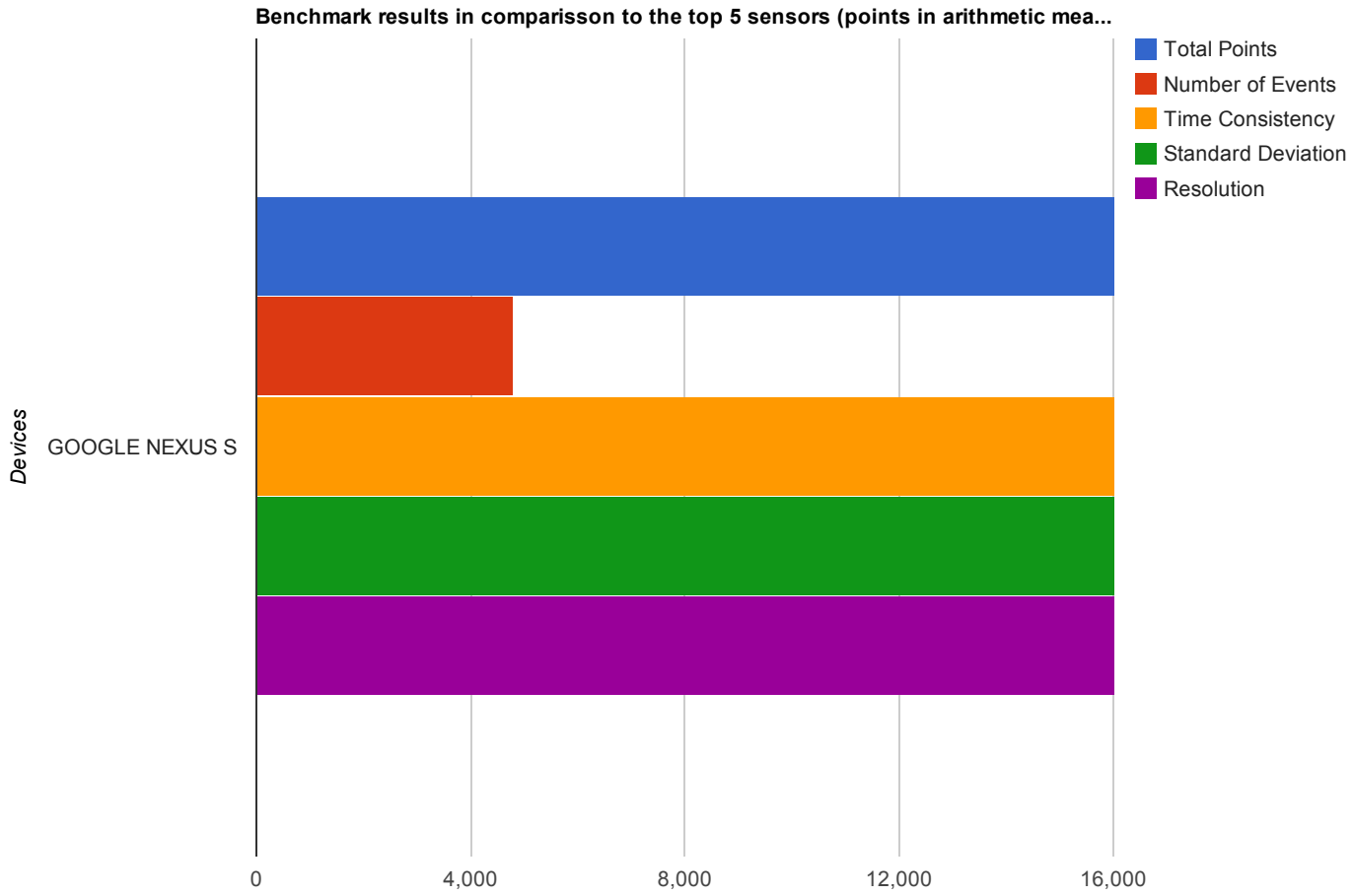
Brand	Android-x86
Manufacturer	Bochs
Device	x86
Model	Bochs
Product	android_x86
Hardware	android_x86
Board	unknown

↑

Sensor Benchmarks

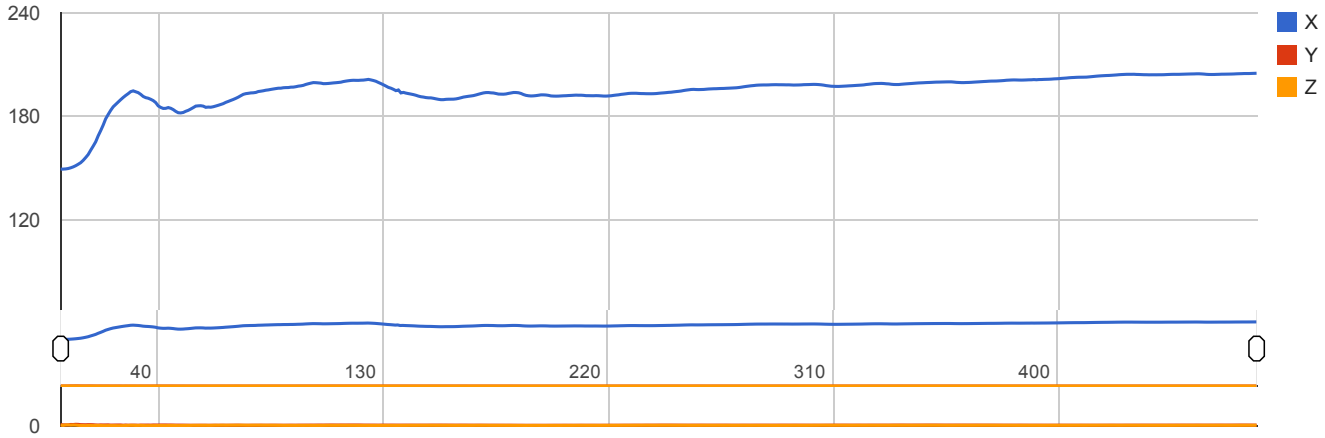
Orientation: Orientation Sensor by Google Inc.

Benchmark



Points

Total	14554
Events	480
Deviation Timepitch	3433
Deviation Values	3074
Resolution	7567

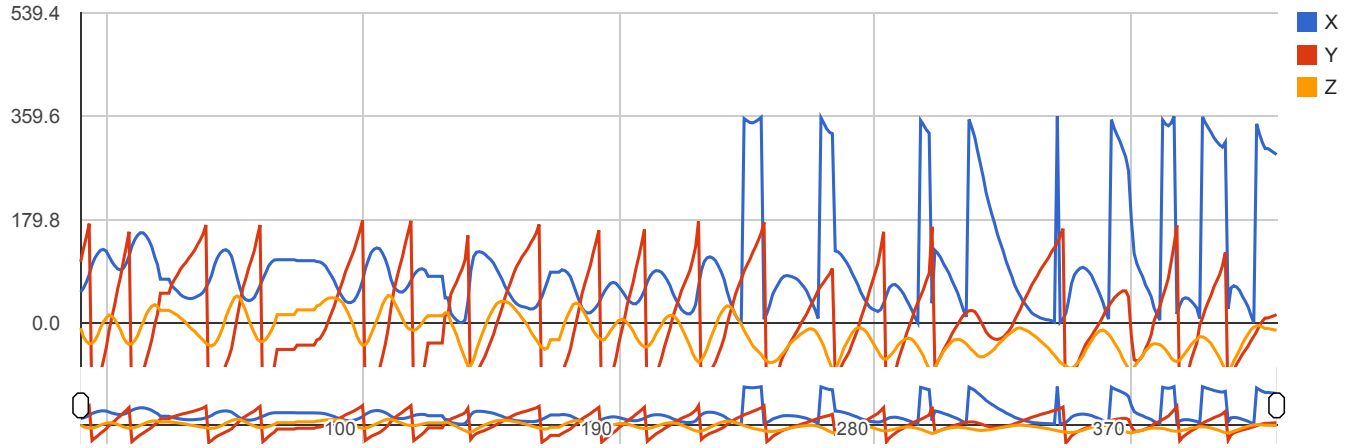


Number of events	480
Timepitch (minimum)	17622000 µs
Timepitch (arithmetic mean)	20061081.4196242 µs
Timepitch (standard deviation)	617548.324171898 µs
Minimum Resolution X	0.0009613037 °
Min. Resolution Y	3.027916e-05 °
Min. Resolution Z	2.0176172e-05 °
Minimum X	149.31502 °
Mean X	195.168065897624 °
Maximum X	205.15082 °
Minimum Y	0.46560785 °
Mean Y	0.566474574866394 °
Maximum Y	1.0428673 °
Minimum Z	0.20484763 °
Mean Z	0.422226777952165 °
Maximum Z	0.66346294 °
Standard Deviation X	9.38154127085435 °
Standard Deviation Y	0.0633600304355813 °
Standard Deviation Z	0.0498417742964972 °
Android Version	4.3
App Version	1.0
App Version Code	9
Battery	0%
Charging	1

SensMark results based on all executed initializations and benchmarks with this specific sensor

Minimum Timepitch	17501000 µs
Minimum Resolution X	3.0517578e-05 °
Min. Resolution Y	2.2172928e-05 °
Min. Resolution Z	5.505979e-06 °
Minimum X	0.1895307 °
Maximum X	359.90366 °
Minimum Y	-179.3778 °
Maximum Y	179.4094 °
Minimum Z	-87.197395 °
Maximum Z	87.90361 °

Initialization



Download all values as JSON array

Number of events	421
Timepitch (minimum)	17500000 µs
Timepitch (arithmetic mean)	23433764.2857143 µs
Timepitch (standard deviation)	38809437.2601905 µs
Minimum Resolution X	0.040836334 °
Min. Resolution Y	0.08538437 °
Min. Resolution Z	0.021385193 °
Minimum X	0.010887903 °
Maximum X	359.65134 °
Minimum Y	-179.77525 °
Maximum Y	178.40813 °
Minimum Z	-89.15096 °
Maximum Z	47.53811 °

About Device

Technical sensor specifications by device

Maximum Range	360 °	
Minimum Delay	20000 µs	The minimum delay allowed between two events in microsecond or zero if this sensor only returns a value when the data it's measuring changes.
Power	13.13 mA	The power in mA used by this sensor while in use.
Resolution	1 °	
Version	1	

Note: In our research we found out that the specifications above do not have to be true.

Device

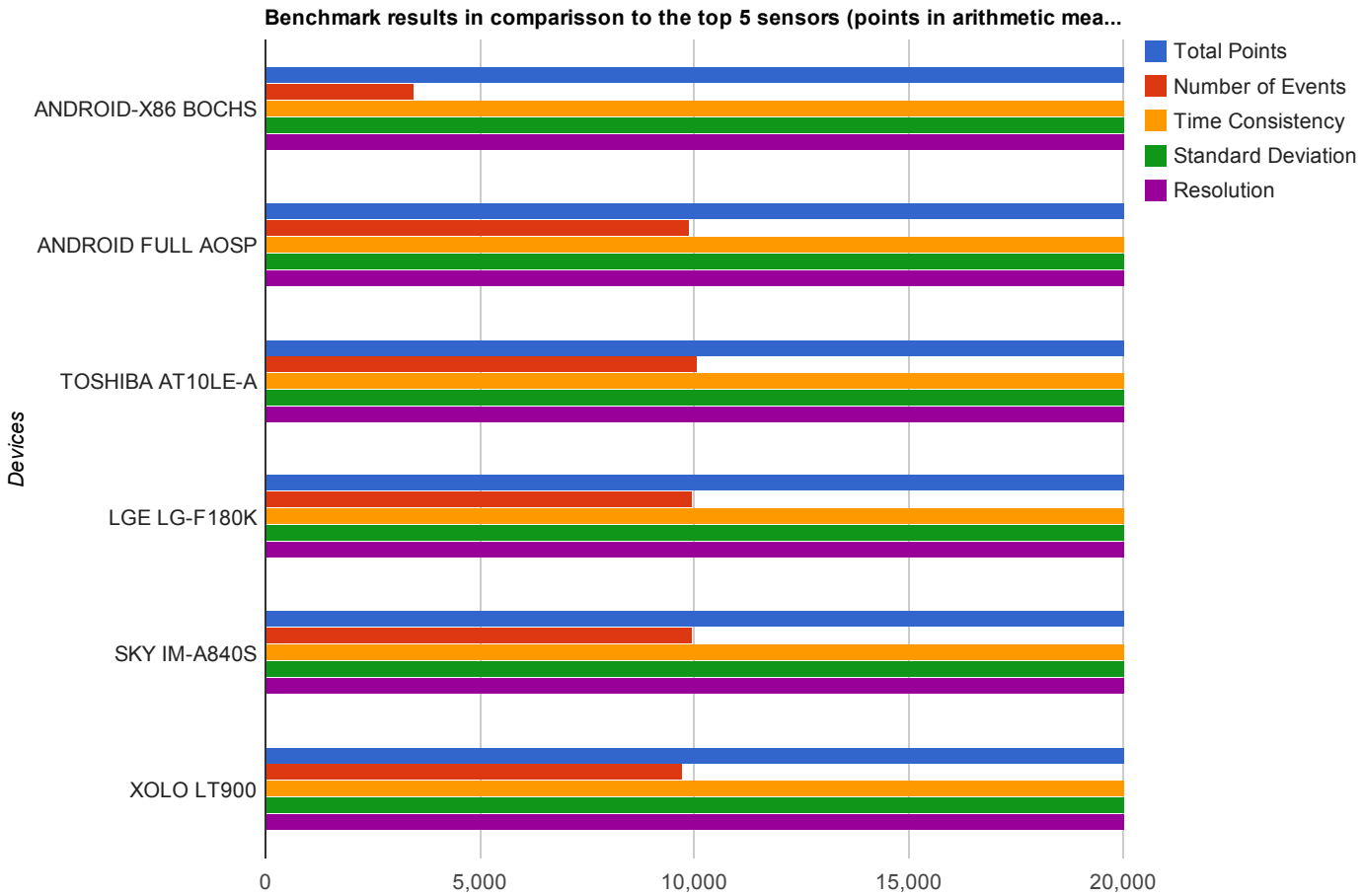
Brand	Google
Manufacturer	Samsung
Device	crespo
Model	Nexus S
Product	soju
Hardware	herring
Board	herring

↑

Sensor Benchmarks

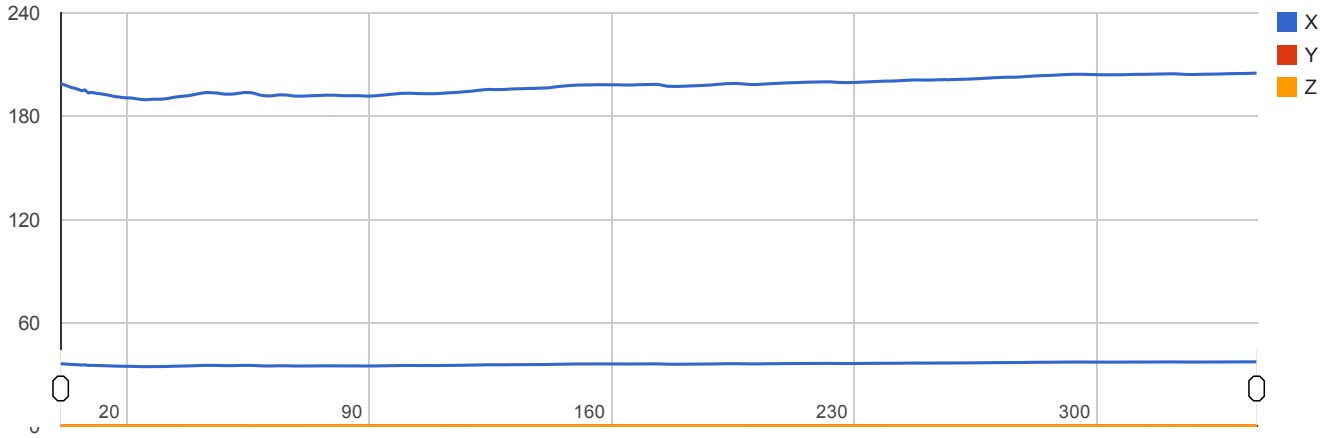
Orientation: Orientation Sensor Emulation!!!! by Columbia University - NYC - Thesis - Raghavan Santhanam

Benchmark



Points

Total	11627
Events	346
Deviation Timepitch	2025
Deviation Values	3388
Resolution	5868



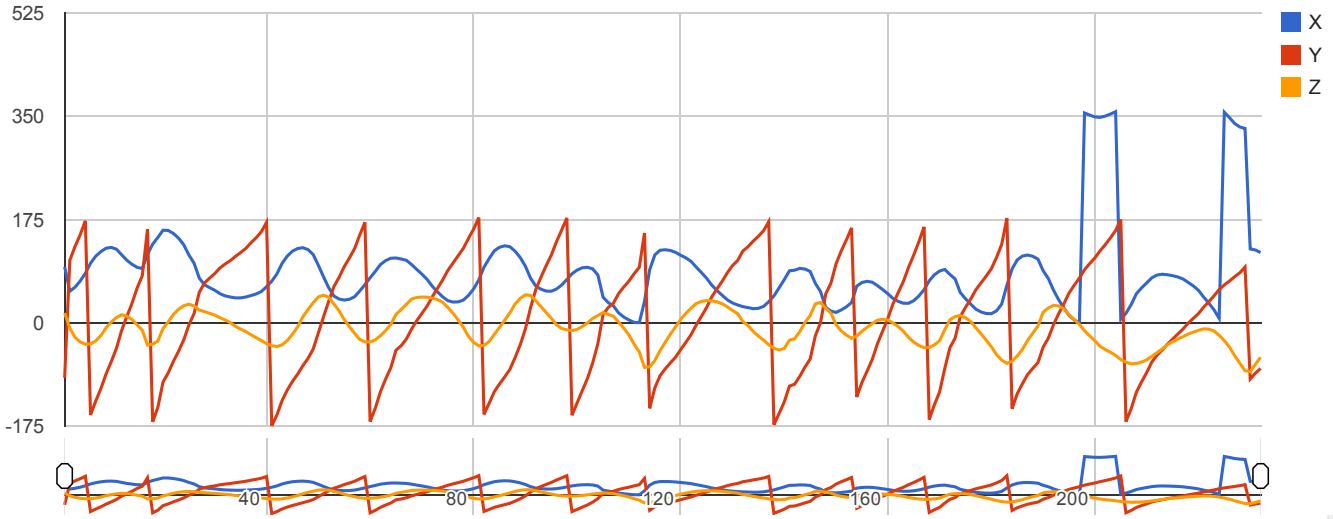
Download all values as JSON array

Number of events	346
Timepitch (minimum)	1093162 µs
Timepitch (arithmetic mean)	20446264.6231884 µs
Timepitch (standard deviation)	16655073.020031 µs
Minimum Resolution X	0.0009613037 °
Min. Resolution Y	6.198883e-05 °
Min. Resolution Z	1.9997358e-05 °
Minimum X	189.699 °
Mean X	197.91206417194 °
Maximum X	205.15082 °
Minimum Y	0.465608 °
Mean Y	0.553732951252447 °
Maximum Y	0.614884 °
Minimum Z	0.366172 °
Mean Z	0.429362644522176 °
Maximum Z	0.475049 °
Standard Deviation X	4.56719400212691 °
Standard Deviation Y	0.0304310834014855 °
Standard Deviation Z	0.0243395266548155 °
Android Version	4.3.1
App Version	1.0
App Version Code	9
Battery	100%
Charging	0

SensMark results based on all executed initializations and benchmarks with this specific sensor

Minimum Timepitch	9223372036854775807 µs
Minimum Resolution X	2147483648 °
Min. Resolution Y	2147483648 °
Min. Resolution Z	2147483648 °
Minimum X	2147483648 °
Maximum X	-2147483648 °
Minimum Y	2147483648 °
Maximum Y	-2147483648 °
Minimum Z	2147483648 °
Maximum Z	-2147483648 °

Initialization



Download all values as JSON array

Number of events	232
Timepitch (minimum)	1098953 µs
Timepitch (arithmetic mean)	43284757.7878788 µs
Timepitch (standard deviation)	225029837.817748 µs
Minimum Resolution X	0.040836334 °
Min. Resolution Y	3.1962662 °
Min. Resolution Z	0.021385193 °
Minimum X	0.010888 °
Maximum X	357.92697 °
Minimum Y	-174.34981 °
Maximum Y	178.40813 °
Minimum Z	-82.66958 °
Maximum Z	47.53811 °

About Device

Technical sensor specifications by device

Maximum Range	360 °	
Minimum Delay	20000 µs	The minimum delay allowed between two events in microsecond or zero if this sensor only returns a value when the data it's measuring changes.
Power	13.13 mA	The power in mA used by this sensor while in use.
Resolution	1 °	
Version	1	

Note: In our research we found out that the specifications above do not have to be true.

Device

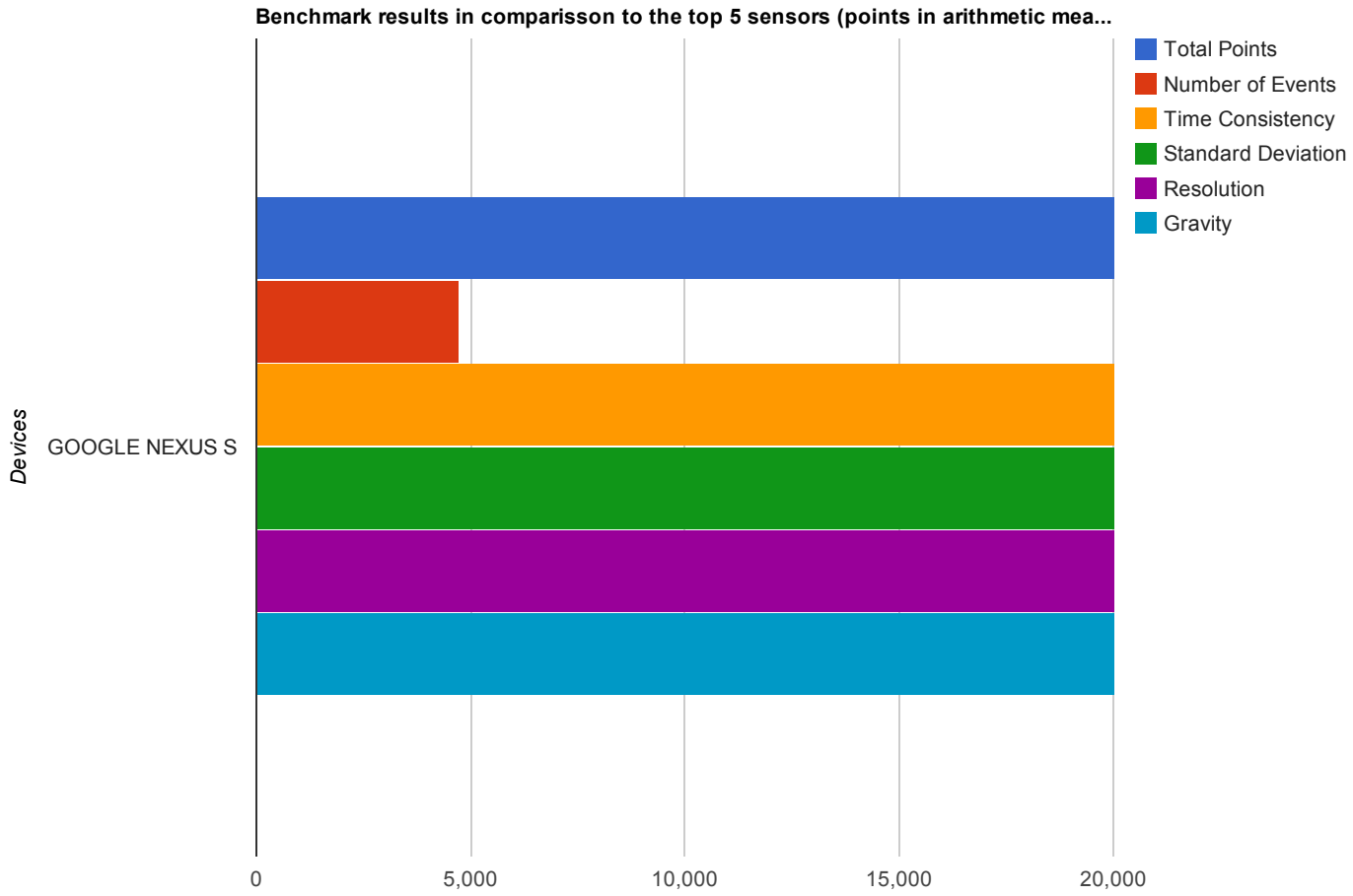
Brand	Android-x86
Manufacturer	Bochs
Device	x86
Model	Bochs
Product	android_x86
Hardware	android_x86
Board	unknown

↑

Sensor Benchmarks

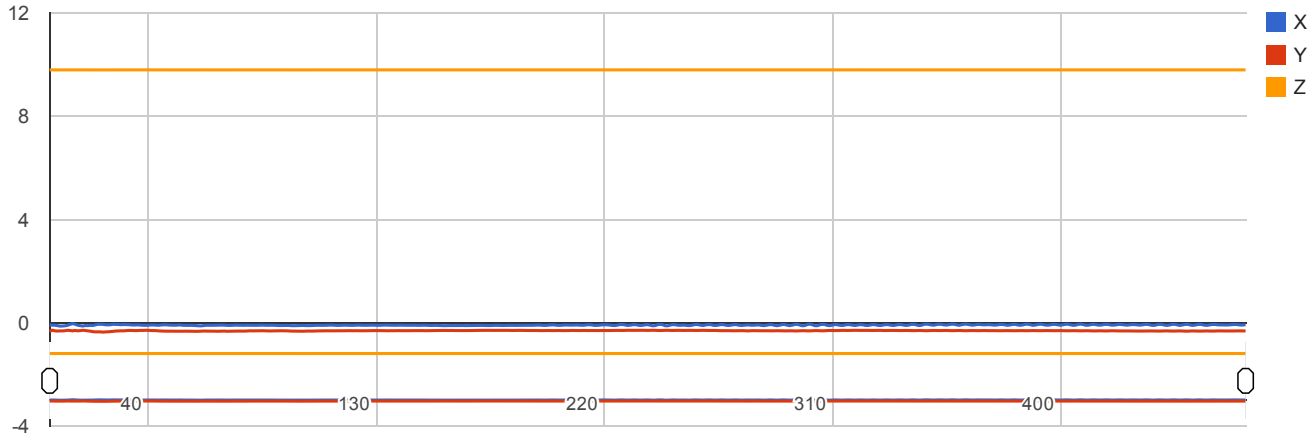
Gravity: Gravity Sensor by Google Inc.

Benchmark



Points

Total	19940
Events	473
Deviation Timepitch	3688
Deviation Values	4983
Resolution	7194
Gravity	3602



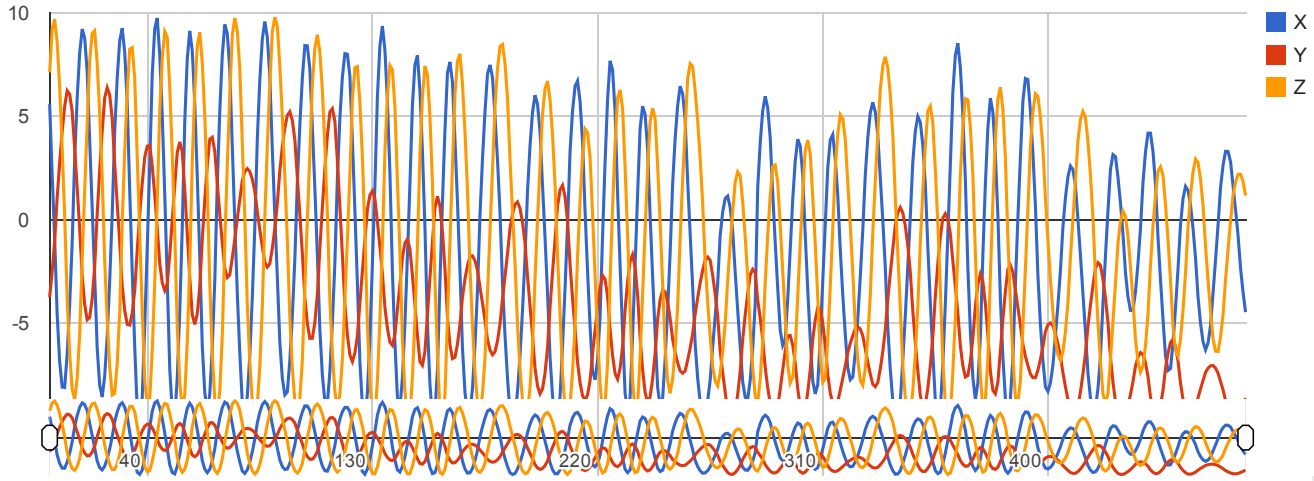
Download all values as JSON array

Number of events	473
Timepitch (minimum)	18288000 µs
Timepitch (arithmetic mean)	20120747.8813559 µs
Timepitch (standard deviation)	487663.603116348 µs
Minimum Resolution X	2.7365983e-05 m/s²
Min. Resolution Y	2.95043e-06 m/s²
Min. Resolution Z	9.536743e-07 m/s²
Minimum X	-0.12489976 m/s²
Mean X	-0.0855280153736881 m/s²
Maximum X	-0.027305666 m/s²
Minimum Y	-0.35864648 m/s²
Mean Y	-0.3048672519829 m/s²
Maximum Y	-0.28636047 m/s²
Minimum Z	9.799911 m/s²
Mean Z	9.80151787973609 m/s²
Maximum Z	9.802134 m/s²
Standard Deviation X	0.0158366919384434 m/s²
Standard Deviation Y	0.0112219425303716 m/s²
Standard Deviation Z	0.000361846781994485 m/s²
Android Version	4.3
App Version	1.0
App Version Code	9
Battery	60%
Charging	1

SensMark results based on all executed initializations and benchmarks with this specific sensor

Minimum Timepitch	17679000 µs
Minimum Resolution X	2.7365983e-05 m/s²
Min. Resolution Y	2.95043e-06 m/s²
Min. Resolution Z	9.536743e-07 m/s²
Minimum X	-9.7954445 m/s²
Maximum X	9.7625 m/s²
Minimum Y	-9.783341 m/s²
Maximum Y	6.3910346 m/s²
Minimum Z	-9.803465 m/s²
Maximum Z	9.802156 m/s²

Initialization



Download all values as JSON array

Number of events	479
Timepitch (minimum)	17679000 µs
Timepitch (arithmetic mean)	20072422.5941422 µs
Timepitch (standard deviation)	1095790.9319448 µs
Minimum Resolution X	0.0016345978 m/s²
Min. Resolution Y	3.6239624e-05 m/s²
Min. Resolution Z	0.007071018 m/s²
Minimum X	-9.7954445 m/s²
Maximum X	9.7625 m/s²
Minimum Y	-9.783341 m/s²
Maximum Y	6.3910346 m/s²
Minimum Z	-9.803465 m/s²
Maximum Z	9.79571 m/s²

About Device

Technical sensor specifications by device

Maximum Range	19.6133 m/s²	
Minimum Delay	20000 µs	The minimum delay allowed between two events in microsecond or zero if this sensor only returns a value when the data it's measuring changes.
Power	13.13 mA	The power in mA used by this sensor while in use.
Resolution	3 m/s²	
Version	3	

Note: In our research we found out that the specifications above do not have to be true.

Device

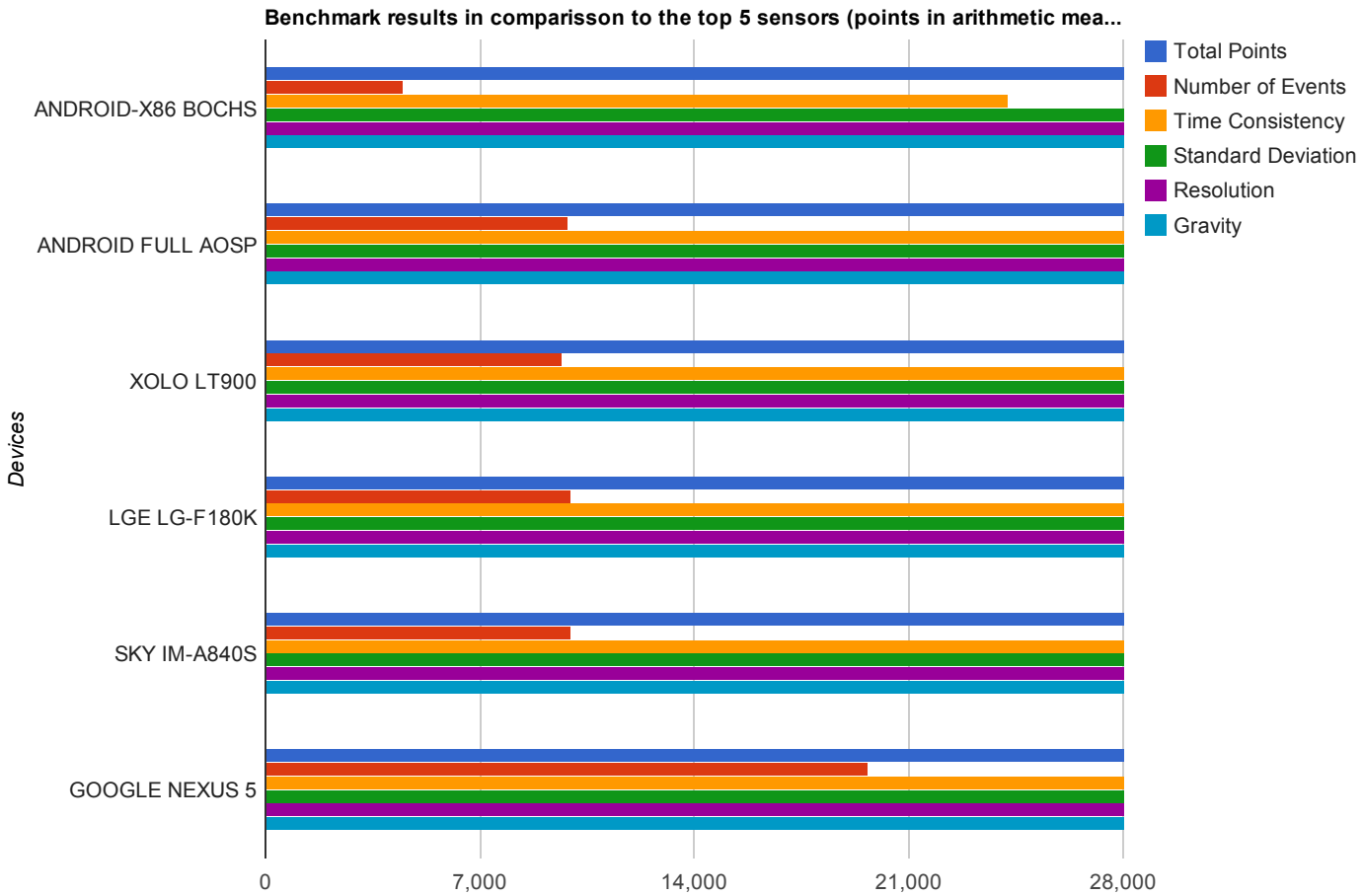
Brand	Google
Manufacturer	Samsung
Device	crespo
Model	Nexus S
Product	soju
Hardware	herring
Board	herring

↑

Sensor Benchmarks

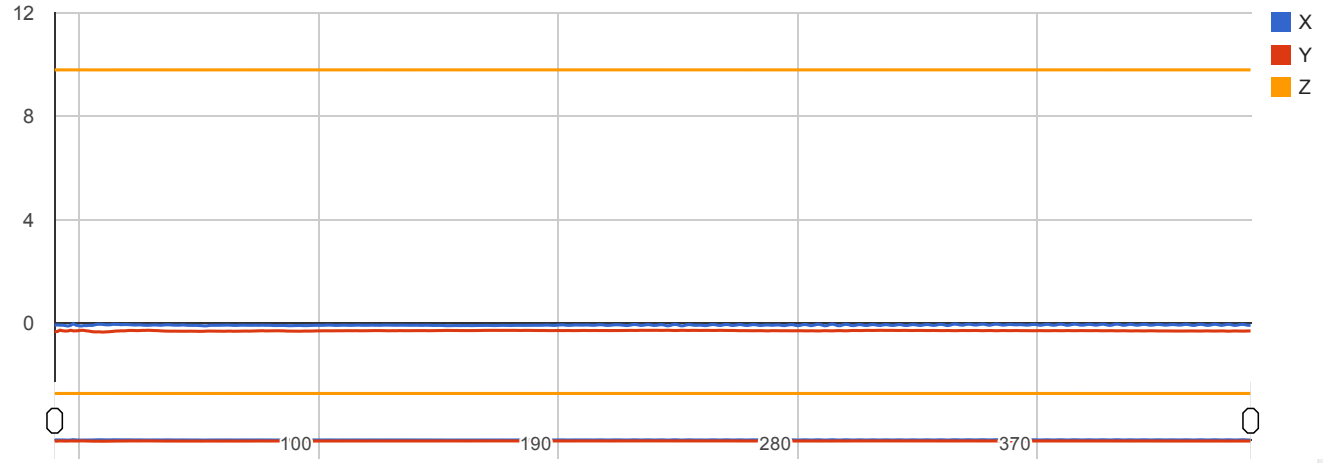
Gravity: Gravity Sensor Emulation!!!! by Columbia University - NYC - Thesis - Raghavan Santhanam

Benchmark



Points

Total	18643
Events	450
Deviation Timepitch	2424
Deviation Values	4973
Resolution	7194
Gravity	3602



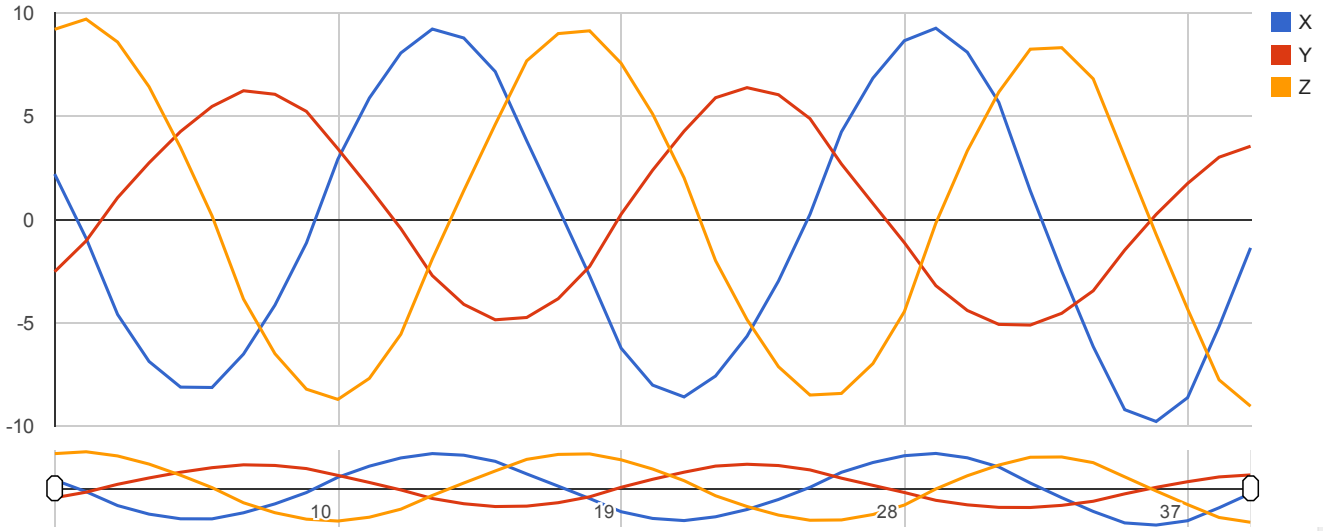
Download all values as JSON array

Number of events	450
Timepitch (minimum)	2147318 µs
Timepitch (arithmetic mean)	22271104.454343 µs
Timepitch (standard deviation)	39262237.2625193 µs
Minimum Resolution X	2.7365983e-05 m/s²
Min. Resolution Y	2.95043e-06 m/s²
Min. Resolution Z	9.536743e-07 m/s²
Minimum X	-0.12353363 m/s²
Mean X	-0.085820348304179 m/s²
Maximum X	-0.027305666 m/s²
Minimum Y	-0.35864648 m/s²
Mean Y	-0.304637827939458 m/s²
Maximum Y	-0.2827893 m/s²
Minimum Z	9.799911 m/s²
Mean Z	9.80152234183417 m/s²
Maximum Z	9.802156 m/s²
Standard Deviation X	0.015740546306606 m/s²
Standard Deviation Y	0.0114616348129756 m/s²
Standard Deviation Z	0.000370016797262376 m/s²
Android Version	4.3.1
App Version	1.0
App Version Code	9
Battery	100%
Charging	0

SensMark results based on all executed initializations and benchmarks with this specific sensor

Minimum Timepitch	2147318 µs
Minimum Resolution X	2.7365983e-05 m/s²
Min. Resolution Y	2.95043e-06 m/s²
Min. Resolution Z	9.536743e-07 m/s²
Minimum X	-9.777154 m/s²
Maximum X	9.268852 m/s²
Minimum Y	-5.1074166 m/s²
Maximum Y	6.3910346 m/s²
Minimum Z	-9.0365095 m/s²
Maximum Z	9.802156 m/s²

Initialization



Download all values as JSON array

Number of events	39
Timepitch (minimum)	2192406 µs
Timepitch (arithmetic mean)	19853208.3947368 µs
Timepitch (standard deviation)	6421929.25196294 µs
Minimum Resolution X	0.01804161 m/s²
Min. Resolution Y	0.033097267 m/s²
Min. Resolution Z	0.07286453 m/s²
Minimum X	-9.777154 m/s²
Maximum X	9.268852 m/s²
Minimum Y	-5.1074166 m/s²
Maximum Y	6.3910346 m/s²
Minimum Z	-9.0365095 m/s²
Maximum Z	9.143451 m/s²

About Device

Technical sensor specifications by device

Maximum Range	19.6133 m/s²	
Minimum Delay	20000 µs	The minimum delay allowed between two events in microsecond or zero if this sensor only returns a value when the data it's measuring changes.
Power	13.13 mA	The power in mA used by this sensor while in use.
Resolution	3 m/s²	
Version	3	

Note: In our research we found out that the specifications above do not have to be true.

Device

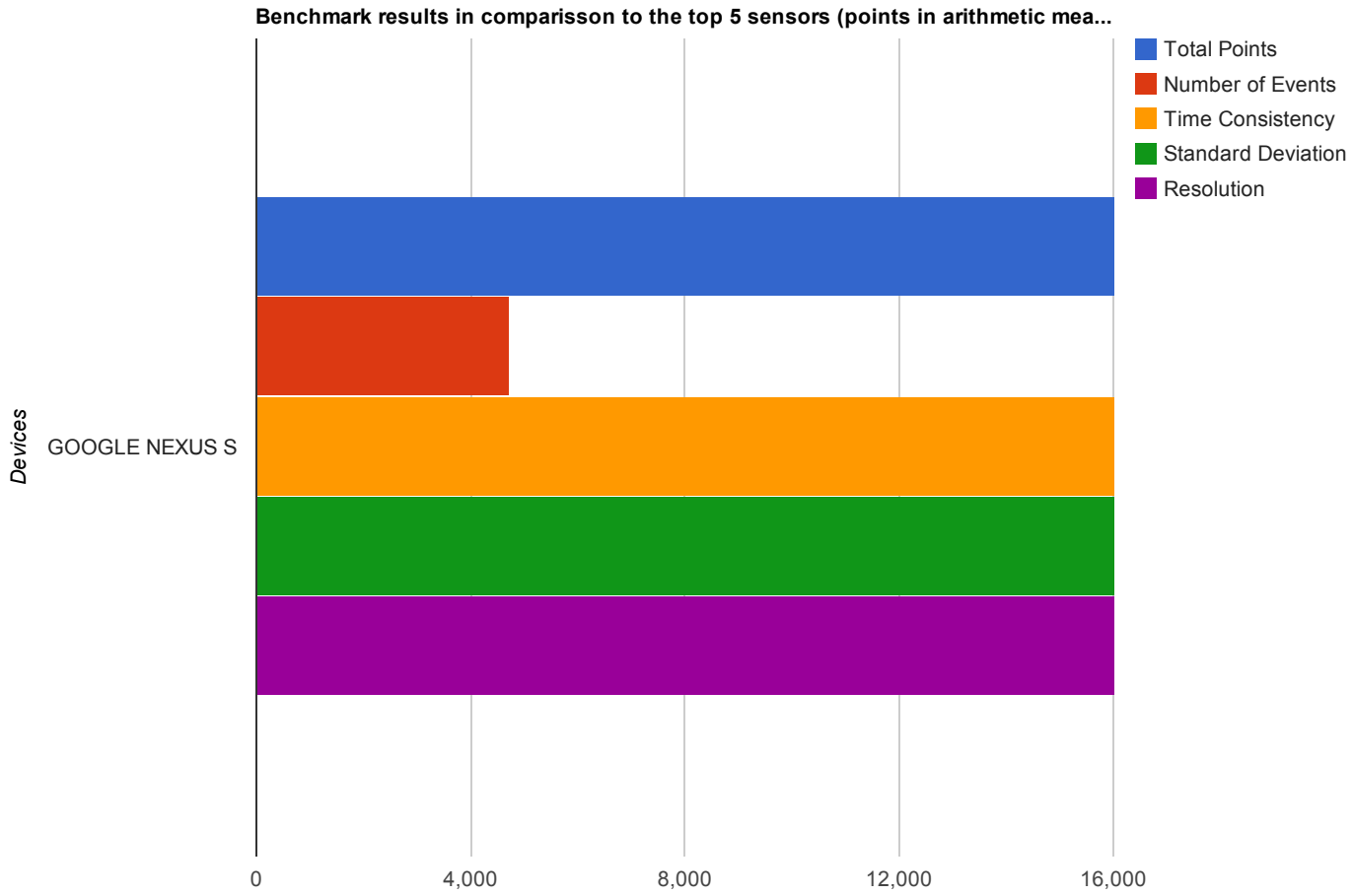
Brand	Android-x86
Manufacturer	Bochs
Device	x86
Model	Bochs
Product	android_x86
Hardware	android_x86
Board	unknown

↑

Sensor Benchmarks

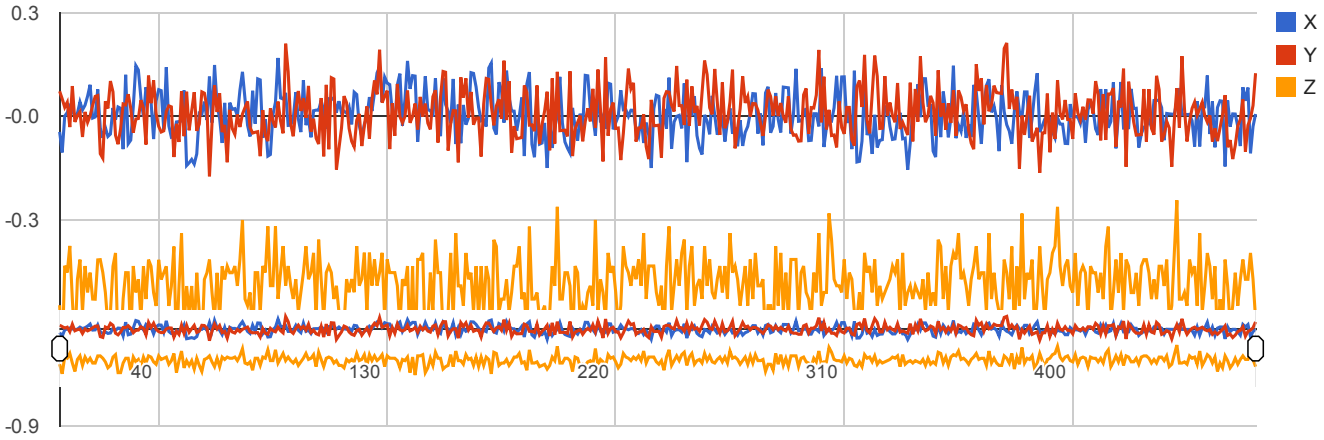
Linear Acceleration: Linear Acceleration Sensor by Google Inc.

Benchmark



Points

Total	14370
Events	472
Deviation Timepitch	3643
Deviation Values	3142
Resolution	7113

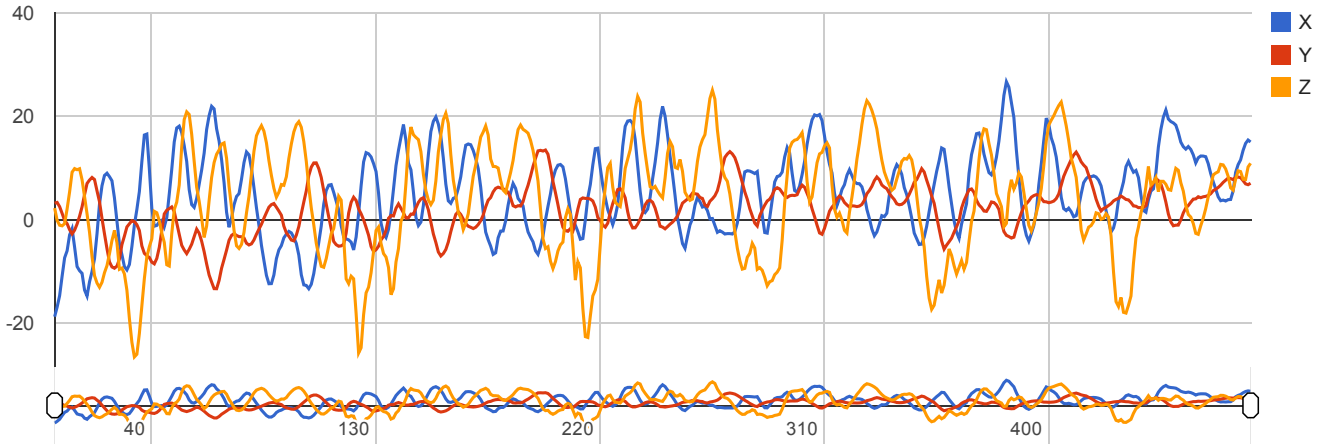


Number of events	472
Timepitch (minimum)	18014000 µs
Timepitch (arithmetic mean)	20089690.0212314 µs
Timepitch (standard deviation)	569211.494073441 µs
Minimum Resolution X	1.148507e-05 m/s²
Min. Resolution Y	2.1636486e-05 m/s²
Min. Resolution Z	9.536743e-07 m/s²
Minimum X	-0.15511778 m/s²
Mean X	0.00486194267395441 m/s²
Maximum X	0.169758 m/s²
Minimum Y	-0.17418107 m/s²
Mean Y	0.00929608593805362 m/s²
Maximum Y	0.2132689 m/s²
Minimum Z	-0.7222843 m/s²
Mean Z	-0.485175845986706 m/s²
Maximum Z	-0.2438364 m/s²
Standard Deviation X	0.0659597691762572 m/s²
Standard Deviation Y	0.0700731261394137 m/s²
Standard Deviation Z	0.0793987900856189 m/s²
Android Version	4.3
App Version	1.0
App Version Code	9
Battery	61%
Charging	1

SensMark results based on all executed initializations and benchmarks with this specific sensor

Minimum Timepitch	17679000 µs
Minimum Resolution X	1.148507e-05 m/s²
Min. Resolution Y	2.1636486e-05 m/s²
Min. Resolution Z	9.536743e-07 m/s²
Minimum X	-18.800388 m/s²
Maximum X	26.709007 m/s²
Minimum Y	-13.391294 m/s²
Maximum Y	13.478722 m/s²
Minimum Z	-26.603035 m/s²
Maximum Z	25.05813 m/s²

Initialization



Download all values as JSON array

Number of events	481
Timepitch (minimum)	17679000 µs
Timepitch (arithmetic mean)	20072020.8333333 µs
Timepitch (standard deviation)	1093518.64488217 µs
Minimum Resolution X	0.0019395351 m/s²
Min. Resolution Y	0.0076088905 m/s²
Min. Resolution Z	0.0051465034 m/s²
Minimum X	-18.800388 m/s²
Maximum X	26.709007 m/s²
Minimum Y	-13.391294 m/s²
Maximum Y	13.478722 m/s²
Minimum Z	-26.603035 m/s²
Maximum Z	25.05813 m/s²

About Device

Technical sensor specifications by device

Maximum Range	19.6133 m/s²	
Minimum Delay	20000 µs	The minimum delay allowed between two events in microsecond or zero if this sensor only returns a value when the data it's measuring changes.
Power	13.13 mA	The power in mA used by this sensor while in use.
Resolution	3 m/s²	
Version	3	

Note: In our research we found out that the specifications above do not have to be true.

Device

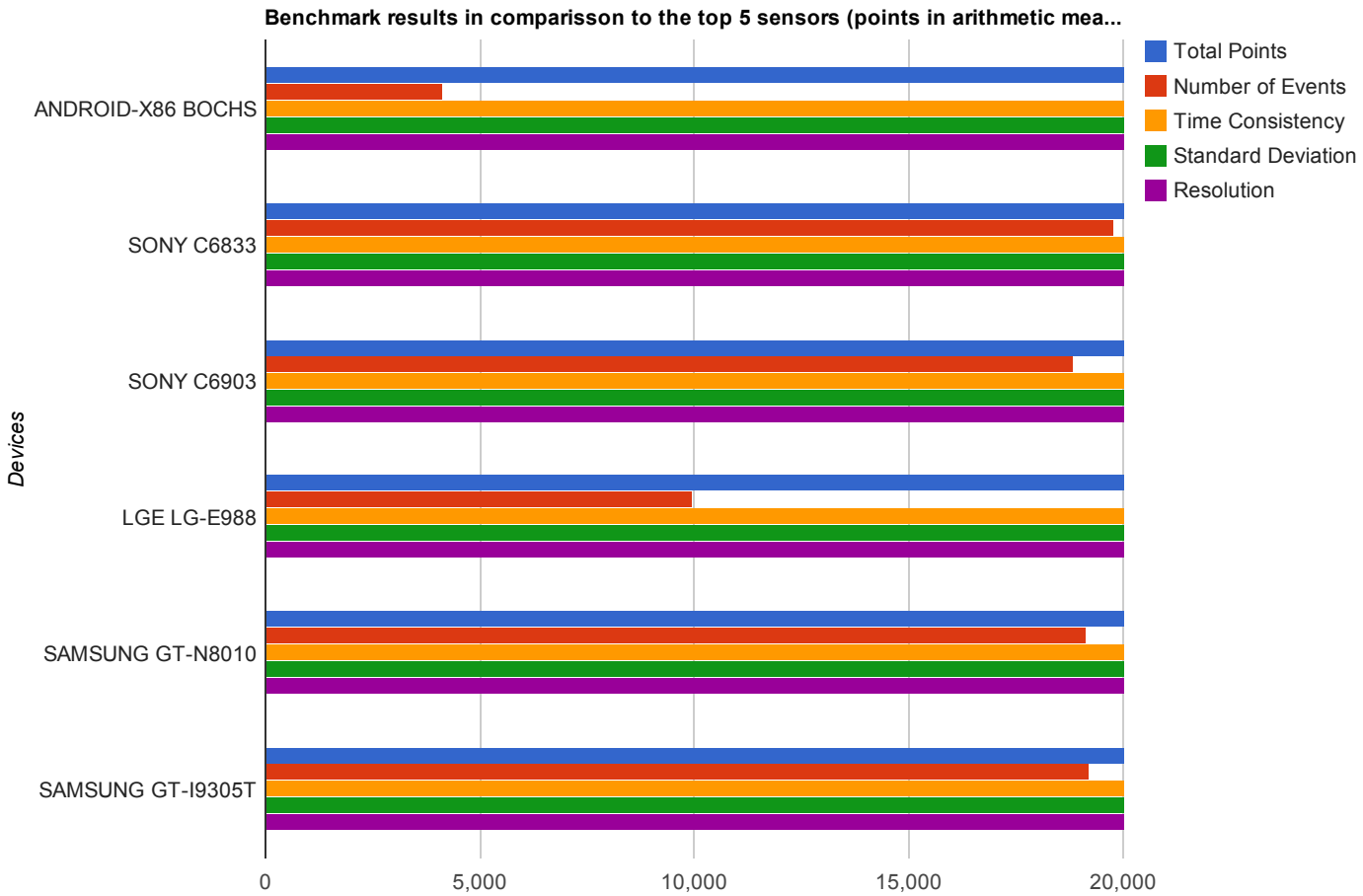
Brand	Google
Manufacturer	Samsung
Device	crespo
Model	Nexus S
Product	soju
Hardware	herring
Board	herring

↑

Sensor Benchmarks

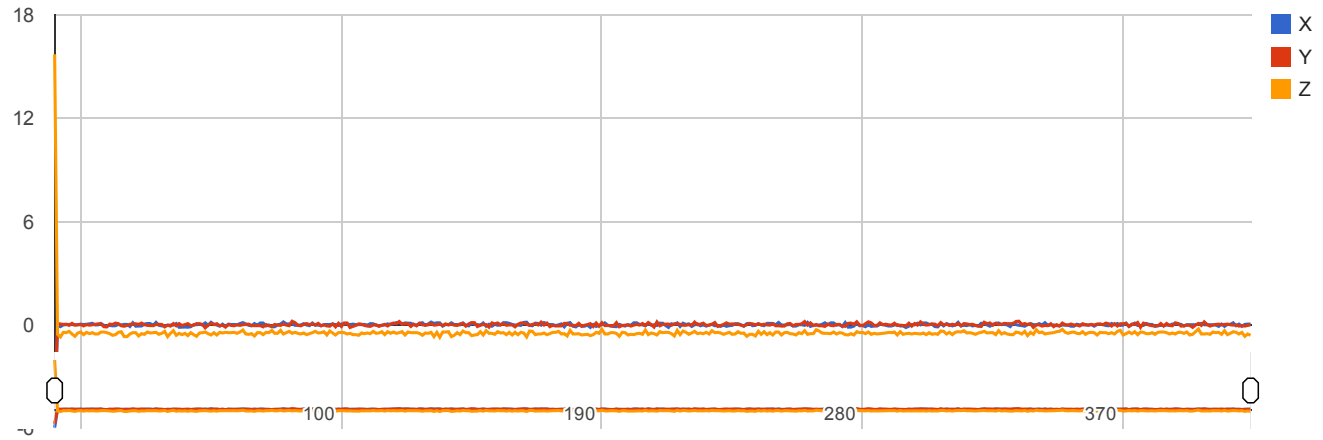
Linear Acceleration: Linear Acceleration Sensor Emulation!!!! by Columbia University - NYC - Thesis - Raghavan Santhanam

Benchmark



Points

Total	12387
Events	414
Deviation Timepitch	2394
Deviation Values	2466
Resolution	7113



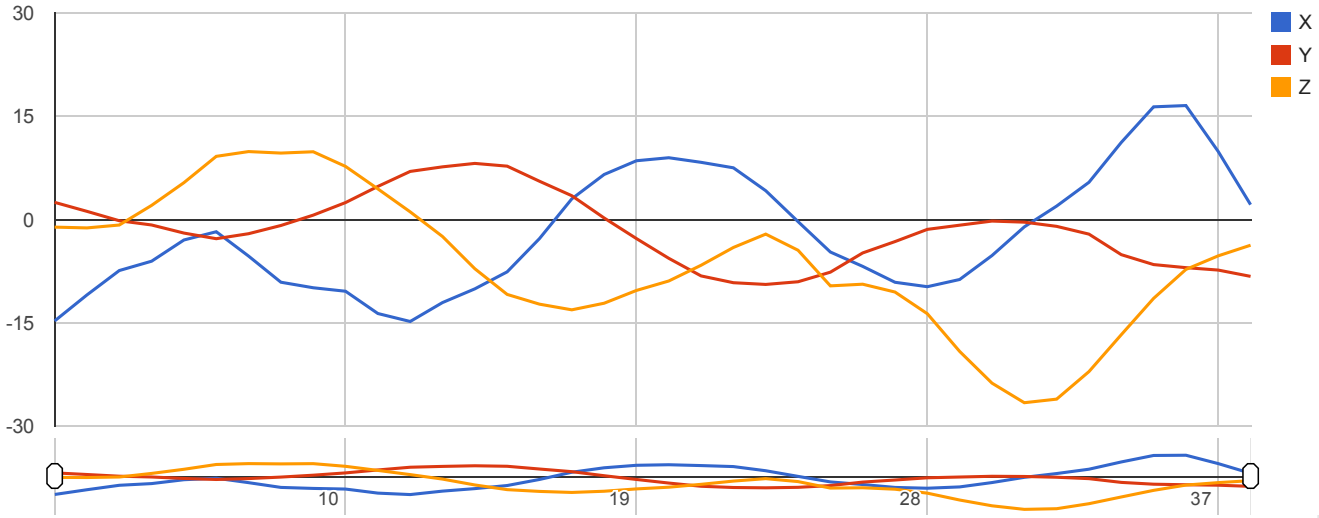
Download all values as JSON array

Number of events	414
Timepitch (minimum)	1083326 µs
Timepitch (arithmetic mean)	24028233.1452785 µs
Timepitch (standard deviation)	66808324.8242716 µs
Minimum Resolution X	1.148507e-05 m/s²
Min. Resolution Y	2.1636486e-05 m/s²
Min. Resolution Z	9.536743e-07 m/s²
Minimum X	-5.8484263 m/s²
Mean X	-0.00794493974887223 m/s²
Maximum X	0.169758 m/s²
Minimum Y	-0.17418107 m/s²
Mean Y	-0.000329550406890126 m/s²
Maximum Y	0.2132689 m/s²
Minimum Z	-0.7222843 m/s²
Mean Z	-0.447017812498526 m/s²
Maximum Z	-0.2438364 m/s²
Standard Deviation X	0.295365793331191 m/s²
Standard Deviation Y	0.23554844226197 m/s²
Standard Deviation Z	0.800892444776628 m/s²
Android Version	4.3.1
App Version	1.0
App Version Code	9
Battery	100%
Charging	0

SensMark results based on all executed initializations and benchmarks with this specific sensor

Minimum Timepitch	1083326 µs
Minimum Resolution X	1.148507e-05 m/s²
Min. Resolution Y	2.1636486e-05 m/s²
Min. Resolution Z	9.536743e-07 m/s²
Minimum X	-14.8026085 m/s²
Maximum X	16.57092 m/s²
Minimum Y	-9.418074 m/s²
Maximum Y	8.165609 m/s²
Minimum Z	-26.603035 m/s²
Maximum Z	15.725853 m/s²

Initialization



Download all values as JSON array

Number of events	38
Timepitch (minimum)	5638455 µs
Timepitch (arithmetic mean)	19931710.027027 µs
Timepitch (standard deviation)	5986690.45291451 µs
Minimum Resolution X	0.18577003 m/s²
Min. Resolution Y	0.15843868 m/s²
Min. Resolution Z	0.1152606 m/s²
Minimum X	-14.8026085 m/s²
Maximum X	16.57092 m/s²
Minimum Y	-9.418074 m/s²
Maximum Y	8.165609 m/s²
Minimum Z	-26.603035 m/s²
Maximum Z	9.884876 m/s²

About Device

Technical sensor specifications by device

Maximum Range	19.6133 m/s²	
Minimum Delay	20000 µs	The minimum delay allowed between two events in microsecond or zero if this sensor only returns a value when the data it's measuring changes.
Power	13.13 mA	The power in mA used by this sensor while in use.
Resolution	3 m/s²	
Version	3	

Note: In our research we found out that the specifications above do not have to be true.

Device

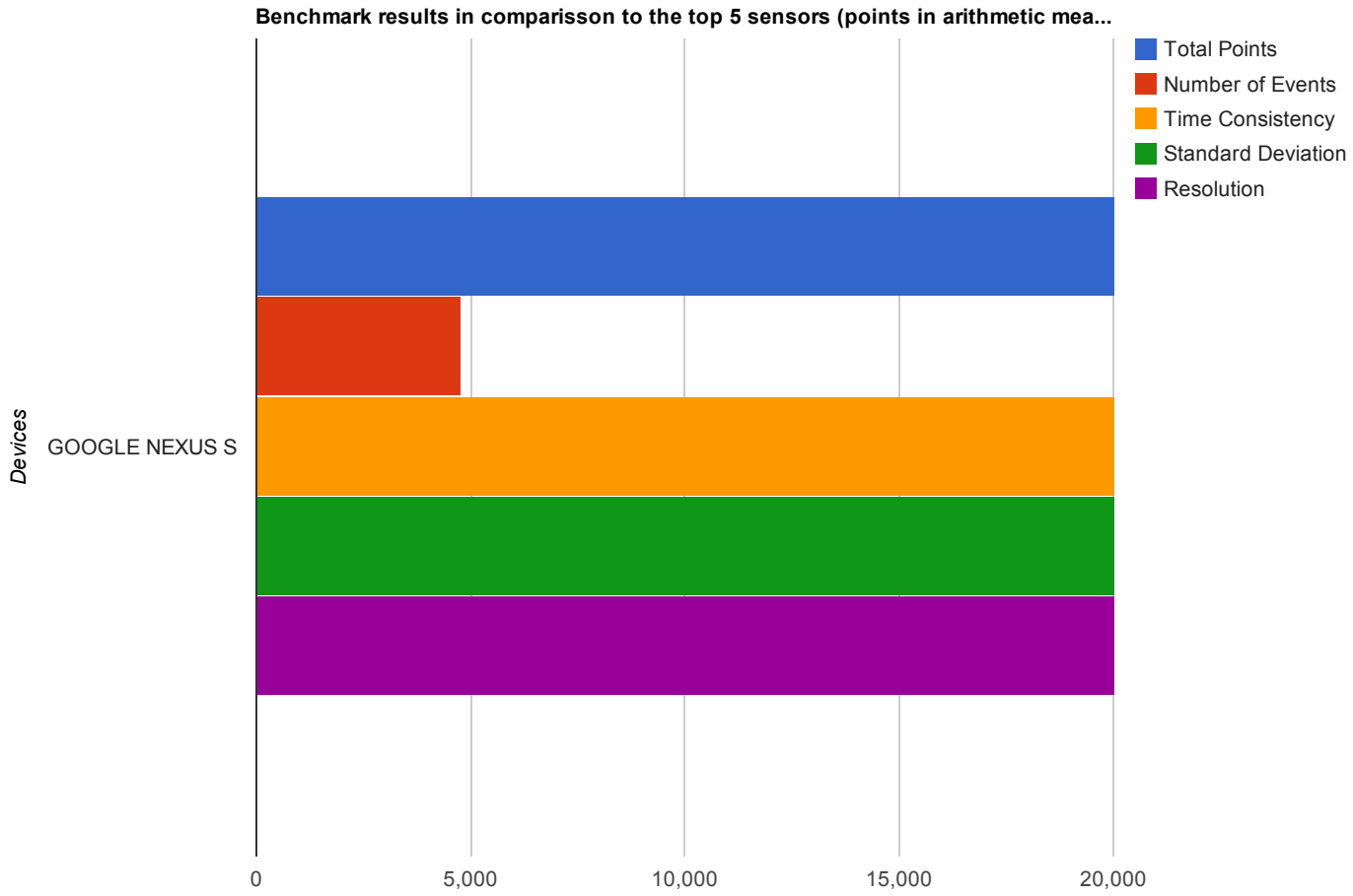
Brand	Android-x86
Manufacturer	Bochs
Device	x86
Model	Bochs
Product	android_x86
Hardware	android_x86
Board	unknown

↑

Sensor Benchmarks

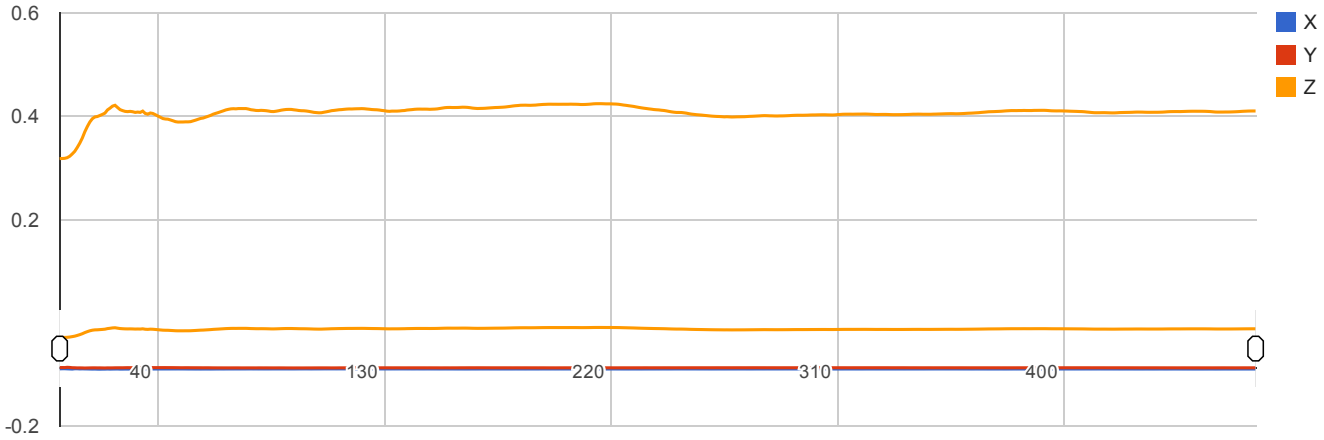
Rotation Vector: Rotation Vector Sensor by Google Inc.

Benchmark



Points

Total	17733
Events	476
Deviation Timepitch	3631
Deviation Values	5064
Resolution	8562

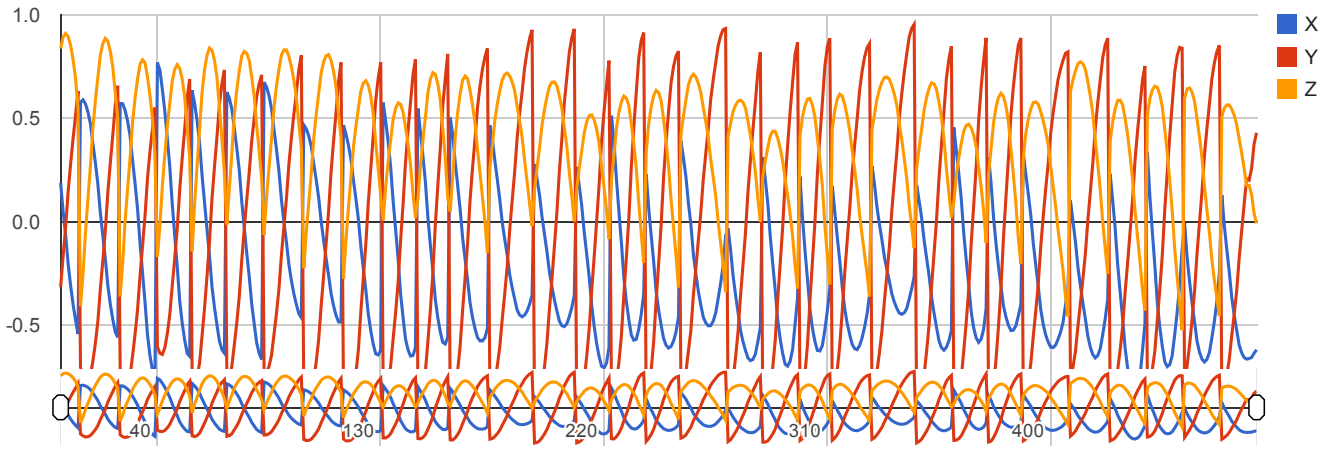


Number of events	476
Timepitch (minimum)	17168000 µs
Timepitch (arithmetic mean)	20051648.4210526 µs
Timepitch (standard deviation)	528809.386918055 µs
Minimum Resolution X	4.0978193e-08
Min. Resolution Y	1.7369166e-07
Min. Resolution Z	3.2782555e-07
Minimum X	-0.019098476
Mean X	-0.016902166526342
Maximum X	-0.014205839
Minimum Y	-0.0068445005
Mean Y	-0.00436856001045676
Maximum Y	0.0007874293
Minimum Z	0.31833655
Mean Z	0.407569030064995
Maximum Z	0.424591
Standard Deviation X	0.000494196573352458
Standard Deviation Y	0.000702911951333481
Standard Deviation Z	0.0137468119591385
Android Version	4.3
App Version	1.0
App Version Code	9
Battery	62%
Charging	1

SensMark results based on all executed initializations and benchmarks with this specific sensor

Minimum Timepitch	17168000 µs
Minimum Resolution X	4.0978193e-08
Min. Resolution Y	1.7369166e-07
Min. Resolution Z	3.2782555e-07
Minimum X	-0.8421365
Maximum X	0.7704579
Minimum Y	-0.9470923
Maximum Y	0.9550614
Minimum Z	-0.52604866
Maximum Z	0.91146535

Initialization



Download all values as JSON array

Number of events	483
Timepitch (minimum)	17679000 µs
Timepitch (arithmetic mean)	20118792.5311203 µs
Timepitch (standard deviation)	1416171.98531898 µs
Minimum Resolution X	0.00040197372
Min. Resolution Y	0.0013104081
Min. Resolution Z	9.11355e-05
Minimum X	-0.8421365
Maximum X	0.7704579
Minimum Y	-0.9470923
Maximum Y	0.9550614
Minimum Z	-0.52604866
Maximum Z	0.91146535

About Device

Technical sensor specifications by device

Maximum Range	1	
Minimum Delay	20000 µs	The minimum delay allowed between two events in microsecond or zero if this sensor only returns a value when the data it's measuring changes.
Power	13.13 mA	The power in mA used by this sensor while in use.
Resolution	3	
Version	3	

Note: In our research we found out that the specifications above do not have to be true.

Device

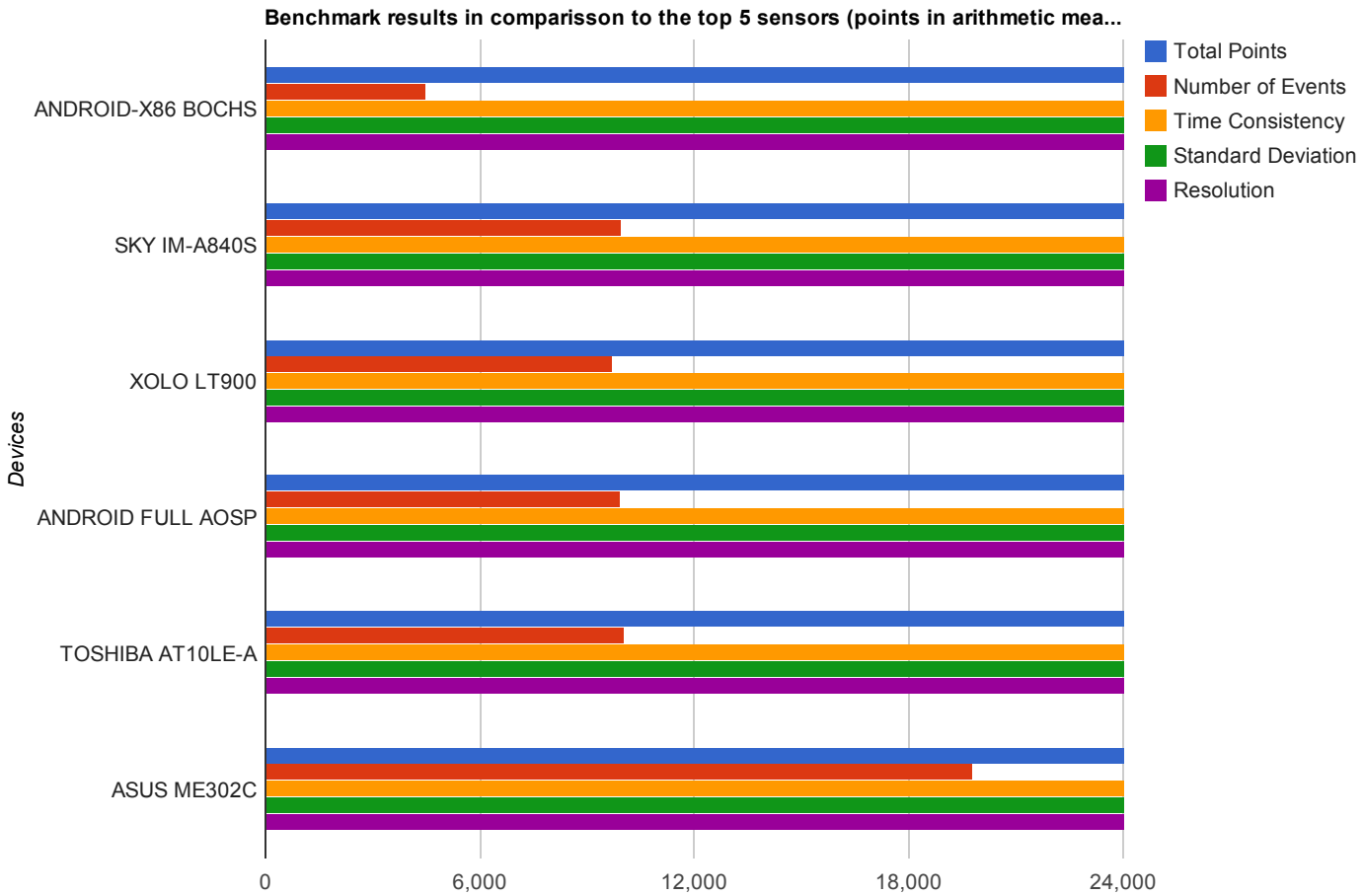
Brand	Google
Manufacturer	Samsung
Device	crespo
Model	Nexus S
Product	soju
Hardware	herring
Board	herring

↑

Sensor Benchmarks

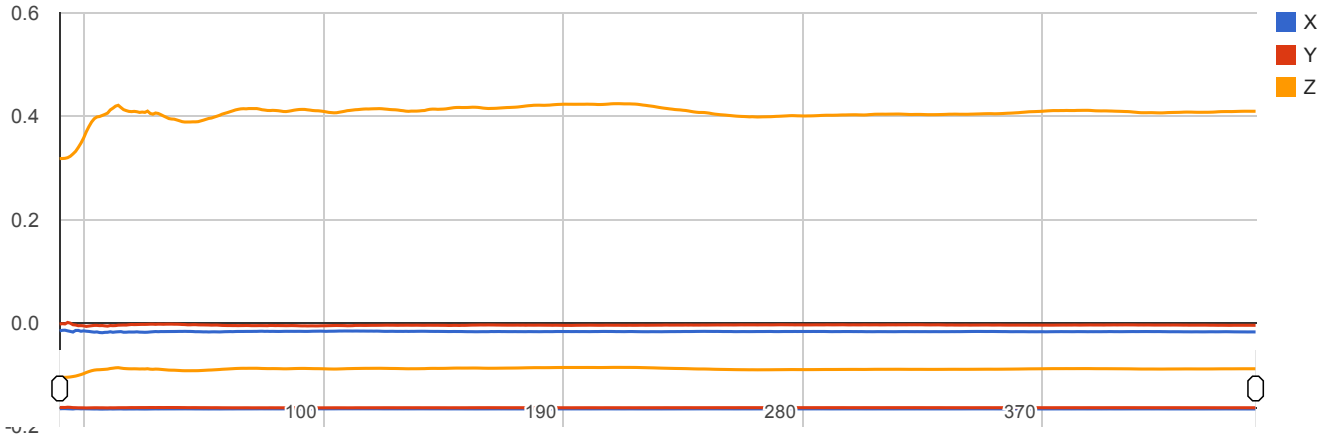
Rotation Vector: Rotation Vector Sensor Emulation!!!! by Columbia University - NYC - Thesis - Raghavan Santhanam

Benchmark



Points

Total	16722
Events	450
Deviation Timepitch	2649
Deviation Values	5061
Resolution	8562



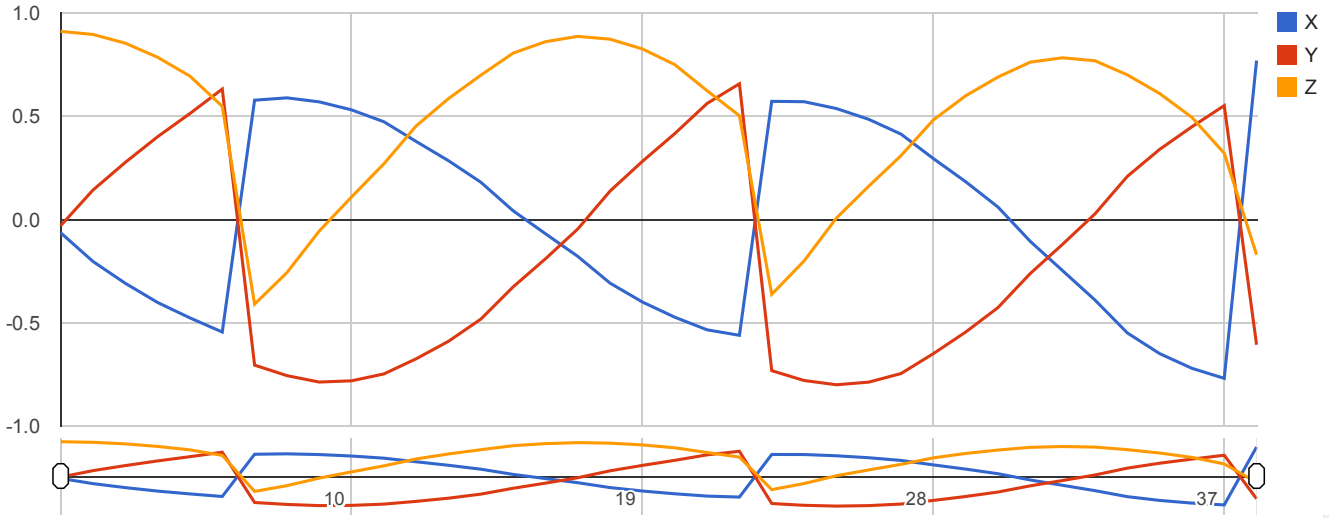
Download all values as JSON array

Number of events	450
Timepitch (minimum)	2195548 µs
Timepitch (arithmetic mean)	20272161.0979955 µs
Timepitch (standard deviation)	10132941.448023 µs
Minimum Resolution X	4.0978193e-08
Min. Resolution Y	1.7415732e-07
Min. Resolution Z	3.2782555e-07
Minimum X	-0.019098476
Mean X	-0.0168825882259343
Maximum X	-0.014205839
Minimum Y	-0.006844501
Mean Y	-0.00434953432659515
Maximum Y	0.000787429
Minimum Z	0.31833655
Mean Z	0.407424929539362
Maximum Z	0.424591
Standard Deviation X	0.000492848744413422
Standard Deviation Y	0.000717955381347782
Standard Deviation Z	0.0141189885758968
Android Version	4.3.1
App Version	1.0
App Version Code	9
Battery	100%
Charging	0

SensMark results based on all executed initializations and benchmarks with this specific sensor

Minimum Timepitch	2195548 µs
Minimum Resolution X	4.0978193e-08
Min. Resolution Y	1.7415732e-07
Min. Resolution Z	3.2782555e-07
Minimum X	-0.7687148
Maximum X	0.7704579
Minimum Y	-0.7998399
Maximum Y	0.6579724
Minimum Z	-0.4093267
Maximum Z	0.8874505

Initialization



Download all values as JSON array

Number of events	38
Timepitch (minimum)	7404753 µs
Timepitch (arithmetic mean)	20031652.9189189 µs
Timepitch (standard deviation)	5700825.88316605 µs
Minimum Resolution X	0.0017365217
Min. Resolution Y	0.0059155226
Min. Resolution Z	0.013722479
Minimum X	-0.7687148
Maximum X	0.7704579
Minimum Y	-0.7998399
Maximum Y	0.6579724
Minimum Z	-0.4093267
Maximum Z	0.8874505

About Device

Technical sensor specifications by device

Maximum Range	1	
Minimum Delay	20000 µs	The minimum delay allowed between two events in microsecond or zero if this sensor only returns a value when the data it's measuring changes.
Power	13.13 mA	The power in mA used by this sensor while in use.
Resolution	3	
Version	3	

Note: In our research we found out that the specifications above do not have to be true.

Device

Brand	Android-x86
Manufacturer	Bochs
Device	x86
Model	Bochs
Product	android_x86
Hardware	android_x86
Board	unknown

↑

Chapter 11
Challenges

This chapter outlines the major challenges that were encountered while shaping up the Sensor Emulation into a work of great quality. This is a ready-reference to the specific solutions chosen for overcoming the challenges encountered and the specific reasons for not being able to overcome some of the challenges.

11.1 Requirements-driven-challenges

11.1.1 Generic

The whole Sensor Emulation was intended to be done at the device driver level. After deep investigation of all the possible alternatives, ease, and portability of the needed Sensor Emulation technique, the HAL specific implementation was finalized. The investigation was an exhaustive one involving the examination of the couple of kernel-level device driver code namely, **<android_src_path>/kernel/drivers/input/misc/adxl34x.c** and **<android_src_path>/ kernel/drivers/staging/iio/accel/lis3l02dq.c**.

11.1.2 Wireless

The challenge of having the Sensor Emulation to remote which means wireless as far as this work is concerned, was faced-off by having socket-communication between the HAL modules be based on TCP/IP to facilitate the correct order of the sensor readings in a reliable way. Thus, the real device being paired can be anywhere in the world as long as it's reachable over IP and also the remote server when used can be running anywhere in this world as long as the host machine running the remote server is reachable over IP.

11.1.3 Lossless and consistent

The real challenge was in getting the RAD sensor readings onto VAD in real-time without any loss over the network. It involved experimenting with many inter-process and inter-thread communication mechanisms as under. Inter-process: 1) File synchronization done in a mutually exclusive manner between two programs(server and client on the host) to communicate sensor readings from one to another. Inter-thread(after merging client and server programs into one and running as threads): 1) Shared-memory access guarded by semaphores. 2) Shared-memory access guarded by futex. 3) Shared-memory access guarded by pipe.

11.1.4 Accurate

The challenge to have the sensor readings reaching the emulator to be accurate was achieved by transmitting the sensor readings in their full numerical precision from the RAD to VAD.

11.1.5 Swift

With respect to the device, the semaphore based notification mechanism employed for letting know the device server that a new sensor reading(more specifically, an individual component of the respective sensor readings which could be triplet, etc), proved to be inefficient and was a hindrance to the real-time nature expected in the Sensor Emulation process. Futex were thought of to be used as they are claimed to be faster user-space mutexes than the generic kernel-based mutexes and semaphores. But, then the pipe usage proved to be the just right choice in terms of the sequentialization of the sensor data for a classic producer(server)-consumer(client) problem. Pipes helped to retain the real-time nature of the readings end-to-end.

11.1.6 Tolerant to High-Frequency Transmission of Sensor Readings

Though the problem of consistency and synchronization got solved, there was a problem with the Qemu emulator that was being used. It was crashing when the readings were sent at high-speed to cater for the real-time nature. The reason became apparent after a detailed analysis of the Qemu logs and searching for the specific error messages in the internet discussion forums. There was a problem in the version of the Qemu(v1.2) in the way of handling the incoming packets in **`slirp/slirp.c`** of Qemu source. The bug causing the problem was fixed later, and using the latest Qemu(v1.6.1 at the time of this writing) which had **`kvm`** code merged from v1.3 onwards, now the readings can be sent virtually any speed - no crash has been experienced thereafter.

11.1.7 Auto-resetting and Auto-restoring

Initially, to keep things simpler, the whole client-server connection setup was being carried out for each and every sensor reading that's being sent and received at all points of the Sensor Emulation. Soon, this proved to be inefficient. This was replaced by on-demand and fault-tolerant connection logic which ensures there is absolutely no activity if there's no connected client in the first place and also if either of the client or the server fail or crash, only then the connection is reset at the failed point and that happens immediately. Some of the servers have a **`SIGPIPE`** handler installed to prevent the entire sensor HAL module from exiting when there is a socket-communication error triggering a **`SIGPIPE`**. Upon handling **`SIGPIPE`**, the state of the respective program(thread) is restored performing a **`longjmp()`** from the previous

successful context saved using **`setjmp()`**.

11.1.8 Optimized in terms of end-to-end socket-communication

Until the port-mapping mechanism was studied and deployed, there was no visual improvement in the sensor readings transmission end-to-end. Then came the concept of the dummy forever-sleep server bridging the gap between the host and guest by simply referring the mapped port on behalf of the respective emulator server.

11.1.9 Rigorously tested

11.1.9.1 Graphics

Another big challenge was to find a 3D racing game for a test and demo of the Sensor Emulation, that uses accelerometer sensor readings while it's running. This game-hunt took almost two whole weeks of tiresome experiment with 100+ games available free for testing from several websites. The real problem that was faced was due to the lack of updated OpenGL ES support on Android-x86. At the time of this writing, only OpenGL ES 1.1 was working on Android-x86 while all the top 3D games would have never been tested in an emulated Android which anyway lacks sensor support, but only real devices which come with OpenGL ES 3.0. So, it took a while to realize that most of the games are ARM based and need ARM emulation in the x86 based Android on Qemu.

11.1.9.2 ARM Translation support

The ARM emulation support library, **`libhoudini`** which was once hosted by formerly active website **`buildroid.org`**(now **`androvm.org`** and no longer hosts these libraries) helped to at least launch some of the ARM based Android games on the emulated Android-x86 which were failing solely for the lack of ARM emulation support, in the first place. However, the binaries of **`libhoudini`** were not available in the Android-x86 source either. So, it took 2-3 days of searching and hacking to find the binaries ultimately in an ISO image hen Android-x86 JB 4.3 ISO was booted, under **`/system/lib`** and **`/system/lib/hw`.**

The lack of the updated OpenGL ES support anyway didn't take the success of having launched the ARM based games any further. So, after few seconds of launch, these games were crashing mostly with a java exception related to mismatch in config specification in one of the graphics related Java sources under Android Framework. Though, there was a workaround announced in some of the websites to a similar error as there was a noted bug in the Android emulator source which was wrongly determining the OpenGL ES version available on the system and was always returning

the old versions to be present which was causing the mismatch in the config specification problem possibly due to the differences between the different versions of OpenGL ES libraries. When the announced fix was incorporated into the corresponding Java sources of Android Framework, that specific issue got fixed but something similar kept cropping up. So, any attempt to fix these were given up as it was learned to be the limitation of Android-x86 itself in not having the latest OpenGL ES version of library capable of running the top 3D Android games which are accelerometer based.

11.1.9.3 Graphics, continued

As an effort to run the advanced 3D games, Hardware(HW) Acceleration was enabled on Qemu for Android-x86 with **`HWACCEL=1`** in the kernel boot parameters. This at least enabled the latest Asphalt 8 Airborne and other 3D games to be fully launched and successfully run as well. But, the UI was way too slow that could have been tolerated. It was learned from the android-x86.org discussion forums that HW Acceleration isn't fully supported on Android-x86 as the related graphics drivers in Android-x86 are not quite functional at the time of this writing.

There is Intel Hardware Accelerated Execution Manager(Intel® HAXM) for Windows hosted Android emulator for the necessary Hardware Acceleration. But, for Linux variant, Ubuntu, KVM was used as informed here.

Other options of emulated Android were tested like Genymotion, etc. As they were not open-source, nothing much helped. The Intel Atom x86 System image was also tried once for running the 3D games with the Android Standard Emulator. There was no difference but the games kept crashing. The reason wasn't investigated further as it was anyway not the problem of the game but was of the outdated graphics support. This was tried with "GPU Support".

GPU passthrough for Qemu was attempted to enable better HW Acceleration. Since, there was only one graphics card in the machine that was being used and there was no secondary machine to connect from, the idea of GPU passthrough was not tried, initially. Also, for better graphics, before finalizing SPICE for Qemu, other VGA options were tried. But, all of them were not as good as SPICE. Hence, SPICEd Qemu is the one that's being used now.

11.1.9.4 WiFi necessity

Apart from the graphics issue, there was a problem in terms of the game data download forced over WiFi. Though the WiFi drivers were loaded and **`wlan0`** and **`wlan1`** interfaces were listed in the emulated Android-x86 environment, they were not coming alive and hence some of the games mandating the game download over WiFi were not tried. Even when the game data was manually placed, they were not considered and download over WiFi was kept on being imposed.

11.1.10 Real

The challenge of making the Sensor Emulation as real as possible, as closer to the behavior of the real hardware sensors, was accomplished by having no modifications in the behavior of the real hardware sensors - their existing sampling frequencies, etc, were left untouched and instead, the emulated sensors' code was fine-tuned to be on par with the high sampling frequencies, etc in addition to having the specifications of the emulated sensors to be exactly same as that of the real hardware sensors like having the power consumption value for the emulated sensor to be same as that of the real hardware sensor, and so on.

11.1.11 Demand-Less

The existing emulator-compatible sensor-based applications can work with the current work of Sensor Emulation without having to be modified at all. This is possible since the current work at HAL level blends so well that it almost becomes a part of the standard Android Sensors framework. All that is being done is to provide the sensor readings from the HAL level instead of from the intended real hardware sensor level. So, the rest of the actions from HAL level onwards until the topmost level where the sensor-based application is running, are exactly same as it would happen in a real device. Hence, the current work is demand-less as far as the emulator-compatible sensor-based application's existing code is concerned.

11.1.12 Independent

The performance of the Sensor Emulation is unaffected by the sensor-based applications running in the virtualized smartphone environment. Regardless of the way in which the sensor-based application is querying the sensor data from the underlying emulated sensors using the Android framework APIs, the Sensor Emulation is going to work independently. More specifically, the rate at which the sensor-based applications are going to query the underlying emulated sensors for sensor data is immaterial to the Sensor Emulation in every sense. The reason being, the Sensor Emulation is happening at the HAL which is independent of what's happening above HAL including the application layer.

11.2 Proving the Generality of Idea

In order to prove the generality of the idea of HAL-based Sensor Emulation, the decision to emulate all the sensors of a real Android Smartphone, Samsung Nexus S, was made. While exercising this decision, there were some obstacles that need to be overcome and are mentioned below.

11.2.1 Additional Sensors in Short Time

One of the last challenges was to emulate all the ten sensors of a real device, Samsung Nexus S in a week's time while whole of the initial work on Sensor Emulation took an entire set of three months in making the emulation as efficient as possible. So, in addition to Accelerometer, all the other sensors were taken up for the emulation and accomplished in a week's time as agreed upon. The obstacle in accomplishing the seemingly mechanically task of emulating the remaining sensors was due the split in the implementation for the real sensors and virtual sensors wherein the virtual sensors are based on Sensor Fusion. The split meant two different libraries - **`libsensorservice.so`** for virtual and **`sensors.herring.so`** for real. Due to the fact that, the Samsung Nexus S had an official release of Android 4.1.2 as its last update and the emulated Android-x86 and its corresponding source was from JB 4.3, for **`libsensorservice.so`**, there was a mismatch in terms of the symbols in the dependent library **`libutils.so`**. It took a span 3-4 days to stop hacking the 4.3 library with dummy methods for the missing symbols from the dependent libraries. It was decided that a custom ROM of Android 4.3 could be flashed onto the same Nexus S and hence there will be no further problems as the both the device and the emulated Android-x86 will be having Android 4.3. With Android 4.3, the missing symbol issue got fixed and the real coding for emulating remaining nine sensors was started and eventually completed.

11.2.2 Specialized applications to test additional sensors

As the emulation of the first sensor, accelerometer was tested with some general sensor-test applications, eventually a specialized accelerometer sensor-based application was necessary and found as well. Similarly, the additional sensors once emulated demanded specialized applications as well. Though not as challenging as others described already, the search for the suitable specialized sensor-based application across the internet and not limited to Android Market, demanded some extra patience in trying out 40+ new sensor-based applications, observing their behavior several times to ensure consistency of sensor readings, and short-listing the applications that work smoothly using which the evaluation of the additional sensors could be carried out. After testing the Sensor Emulation with the basic sensor-test applications and advanced sensor-based 3D games, the overall Sensor Emulation was benchmarked using a Sensor Benchmarks evaluator application. Details of these specialized applications and the data collected in using them are mentioned in the evaluation section.

11.3 Inevitable Problems

Among the miscellaneous but inevitable problems challenging the work, there were quite a few. 1) The very initial troublesome download of the entire Android-x86 JB 4.3 source from android-x86.org. The trouble was due to the intermittent network failures on the download servers forcing repeated downloads

which almost appeared never-ending until after a week when it was complete with no syncing issues. To add, initially the attempts were made with number of syncing jobs to be one but later to save time and speed up the work, more than one was started to be used. 2) Insufficient disk space problems preventing full downloads which got fixed after some cleanup. 3) While trying to flash Android 4.3 on the real device, Samsung Nexus S, two of such devices got seemingly damaged and now they are not even switching on. Reason for the same is still not clear as they don't even go to fastboot mode. So, the third device is what being now used with utmost care. 4) The machine(laptop) being used for the entire work had its some of the keyboard keys stopped from working needing an external keyboard, all of a sudden, for which an external keyboard for the rest of the work. 5) The laptop's screen came off needing to tape it together with the base of the laptop.

11.4 Human Factors

Last but not the least, the endless determination, energy, and passion to complete the entire Sensor Emulation work in a span of approximately three months from September 2013 to November 2013, while the work was targeted for a total of two semesters from Spring 2013 to Fall 2013, single-handedly.

Chapter 12
Usability Factor

Usability of the current work of Sensor Emulation by sensor-based applications has been studied with the following factors in mind.

12.1 Graphical Dependencies

Any Android sensor-based application can make use of the Sensor Emulation as long as the sensor-based application is compatible with emulated Android-x86, specifically in terms of graphical dependencies. All the possible issues concerning graphics are discussed in the "Overcoming Challenges" chapter in detail which could prevent an affected sensor-based application from using the Sensor Emulation altogether. The application might just fail to launch itself. However, with proper graphics support either with updated OpenGL ES library or with a working Hardware Acceleration in Android-x86(as it still has bugs as discussed in the "Overcoming Challenges" section) and/or with GPU-passthrough mechanism, those graphics-dependent sensor-based applications failing to launch or crashing after launching or executing slow once launched, will be able to use the sensor readings from the emulated sensors.

Having said that graphical dependency issues of a sensor-based application can prevent it from using the current work of Sensor Emulation as in such a case the application won't even launch or may crash after launch, it should be clear that the graphics and sensors have absolutely no relation with each other and hence, so are the graphical dependencies of a sensor-based application and the current work of Sensor Emulation.

12.2 Time Delays

Maximum reported time delay of 0.6 seconds for the Sensor Emulation shouldn't be a matter of concern given that there will be anyway a minimum inevitable delay in the Sensor Emulation. The minimum inevitable delays are due to mainly three factors:

1. Network overhead involved in favoring one of the main strengths of the current work which is remoteness.
2. The virtual inherent limit on the number of sensor readings(bytes) incoming that can be processed by the emulator of Android-x86(Qemu in the current work).
3. The necessity of having a minute sleep interval between the sock-receive calls within the innermost loop of the sensor servers on the emulator receiving the respective sensors' readings. And this necessity is important as it prevents the

sensor servers from hogging the CPU making the whole Android-x96 UI appear sluggish and hence unacceptable.

Hence, with a delay as moderate as 0.6 seconds, the current Sensor Emulation work can safely be used to serve in real-time and remotely as well.

12.3 Multiple sensors simultaneously

Usage of more than one sensor at the same time might lead to a slight increase in the time taken for the sensor readings for a particular sensor to reach from the real phone to virtual phone due to number of factors such as increased network bandwidth consumption as each emulated sensor would maintain a dedicated network connection with the respective real hardware sensor(between the respective Sensors HAL modules, to be specific) for retrieving the real hardware sensor readings. So, there will be as many TCP/IP network connections as the number of emulated sensors that are active. For the interested, there are lot of other minor factors that affect the time taken in a small extent and these have been discussed in a more detailed manner in one of the previous chapters called “Evaluation”. However, it should be clear that these are inevitable delays when there are a number of stringent requirements to be met at the same time - remoteness, real-timeness, speed, accuracy, and a lot more. Fortunately, these inevitable delays are negligible as well. So, the current work does fare well even when more than one emulated sensor is in use.

Chapter 13
Related Works

There is no other system level Sensor Emulation for Android which is crucial in making the emulation much closer to a real hardware-driven sensor environment.

13.1 Android Emulators

Below are the works related to Android Emulators being discussed for their sensor support or the absence of.

SensorSimulator - A userspace Android application providing artificially generated sensor readings based on input devices etc. However, this has no way of pairing virtual Android device with a real Android device in **real-time**, although a record-replay facility has been recently added. Any application requiring the readings from this simulation needs to be modified so that it contact this stand-alone sensor simulator stand-alone application to get the readings being generated. Not sure if this is compatible with Android-x86 as well[9]

SensorEmulation is said to work but is a wired approach based on USB connectivity. It can't be done in a remote manner, say over the network using WiFi. Moreover, according to the lead developer of that project in the respective GoogleOnlineDiscussionGroup, there hasn't been any work from long time, there will be a little bit of delay in the emulation and also they haven't measured anything. In addition, this sensor emulation work only emulates accelerometer and compass. Not sure if this is compatible with Android-x86 as well.[10]

SamsungSensorSimulator is a Windows-Mac-only solution does simulate all the sensors that are emulated by the current work and some more as well, with a linked(paired) device over WiFi similar to what the current work does in terms of pairing a real Android device, as explained throughout this document. However, after reading its documentation in detail, below are the conclusions made though questions on these have been asked in its respective Samsung discussion forum and other Q&A sites as well and answers are being awaited from quite a while ago.

1. First of all, the documentation doesn't mention if there is any possibility of a sensor-based application using those sensor readings(both artificially generated and that of a linked device). Second of all, if it's really possible for any

sensor-based application to use these sensor readings in some way, it's not clear if it can be in real-time.
2. It mandates the Android version to be 2.2.
3. All the documentation says that is, it helps in visualizing a real linked device's sensor behavior on the PC, apart from simulating the sensor readings without any linked device. The visualization requires the Samsung Sensor Simulator plugin to be installed for Eclipse IDE and inside the Eclipse IDE, one can see the visualization of these sensor readings with some controls as well. This visualization is made possible by running SensorReplay(.apk) on the real Android device which tries to connect to SimulatorAndroidDevice(.apk) running on the emulator using a pre-agreed port and the ip address of the host running the emulator.
4. The simulator is mentioned to be installable with a set of requirements about the platform, etc which includes the OS requirements to be Windows XP sp2, Windows 7, or Mac OS 10.6 which means no Linux-based support.
5. No information on how fast the simulation will be when using a linked device.
6. If at all any sensor-based application running on the emulator can use the sensor readings of a linked device, it's not clear whether that sensor-based application's code needs to be modified to include some extra specific code to interact with some module of the simulator running within the emulator to get the sensor readings.
7. If possible for any sensor-based application on the emulator to use the sensor readings from a linked device, it's not clear whether more than one such sensor-based application on the emulator to use the same sensor readings at the same time in real-time.
8. Not clear whether applicable to Android-x86 as well.[11]

Samsung GALAXY Tab Emulator - mandates Android-2.2 with API 8 revision 1. Limited to emulating Samsung Galaxy Tab and hence only its sensors which are gyroscope, geo-magnetic sensor, accelerometer, and light sensor. Below are the immediate questions that one would have.

1. How are these sensors being emulated? It's not documented at all.
2. Whether linking(pairing) with a real device is supported or not - not specified anywhere in its webpage. And if supported, does the Sensor Emulation happen in real-time with remoteness being supported for linking the real device?
3. Can the sensor-based applications running on the emulator use these sensor readings in real-time? If yes, whether the applications need to be modified for getting these readings and so on.
4. Again, not sure whether, this is compatible with Android-x86 as well, at least Android-x86 2.2.

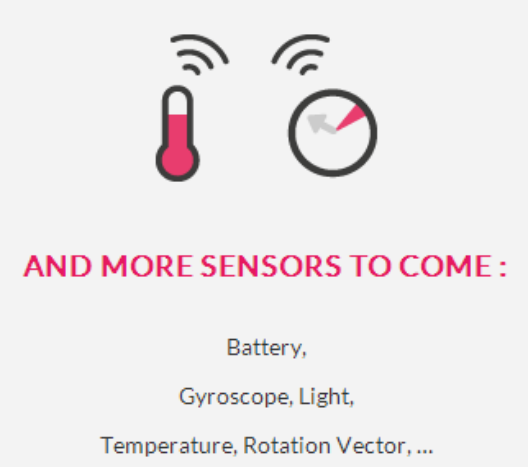
As it may be obvious, the above set of questions applicable for Samsung Sensor Simulator also apply for this Samsung Galaxy Tab Simulator as well.[12]

Standard Android Emulator lacks Sensor support altogether as mentioned below which is taken from its online documentation.[13]

> **Don't test your code on the emulator**
>
> You currently can't test sensor code on the emulator because the emulator cannot emulate sensors. You must test your sensor code on a physical device. There are, however, sensor simulators that you can use to simulate sensor output.

Manymo, claimed to be a better emulator that's in-browser. However, there is no information on the sensors being "simulated" and also not "emulated", as there appears no concept of linking(pairing) a real device with the virtual device run by Manymo. In spite of this, the pre-installed API demo's sensors demo application was launched and only below stuff was seen as far as the information on available sensors is concerned. With no sensor names being displayed, it's really not clear as to what sensors do these indicate -- the immediate guesses would be Orientation sensor being "simulated" and not "emulated" as said earlier. As usual, no information on Android-x86 support.[14]

Genymotion, a Multi-OS solution claimed to be fastest/faster Android emulator available just simulates(and not emulates!) very few sensors such as GPS, Rotation, etc but yet to support other major sensors such as Gyroscope, etc as mentioned below which is taken from its features webpage. However, it seems like going to be only "simulation" without having the possibility to use real hardware sensor readings at all. As usual, no information on Android-x86 support.[15]



Bluestacks, a Windows-Mac only solution supports playing mobile games on PC including Android ones. However, there is no much information about the sensors being supported in the form of "simulation" at least. Some of the internet discussions on stackoverflow.com, and other blogs(one is this) state that accelerometer is being simulated with the keyboard inputs. As usual, no information on Android-x86 support.[16]

Jar Of Beans, a Windows-only solution, was a Android Emulator Portable x86 and it has been discontinued by its developer due to some reasons related to adware, etc. Yet there is no information on whether it supported at least simulation of sensors if not emulation. But, it indeed supported Android-x86 as opposed to the ones mentioned above. If one wants to try out this discontinued project's old code, here are they hosted.[17]

YouWave for Android - Android on Windows PC as the name says is a windows-only solution. Most importantly, there is no sensor support at all in addition to some other limitations which are seen in the below snapshot from its download page. And it's not free but has some free trial period. And it's not mentioned whether Android-x86 is supported or not.[18]

13.2 Works on Mobile Virtualization

Cells: A Virtual Mobile Smartphone Architecture is a virtualization architecture for enabling multiple virtual smartphones to run simultaneously on the same physical cellphone in an isolated, secure manner. This work allows access to the underlying real hardware components of a physical cellphone for the multiple virtual smartphones running on the same physical cellphone including sensors. But, this work doesn't emulate sensors at all but only allows the access to the existing real hardware sensors. Hence, this work bears little relation with the current work due to the mobile virtualization common topic but the relation ends there.[19, 20]

VMware Horizon Mobile, formerly known as VMware Horizon Mobile Virtualization Platform and VMware Mobile Virtualization Platform, in that order, is a solution to multiplex the real hardware(including sensors) between the host and guest running on the same real smartphone. So, there is no aspect of Sensor Emulation happening in a remote manner and in real-time.[21]

Clouds on the Academic Horizon talks about using virtualization technique in splitting of a single physical piece of hardware into independent, self governed environments, which can be scaled in terms of CPU, RAM, Disk and other elements. No emulation of sensors has been discussed.[22]

Remote mobile test system: a mobile phone cloud for application testing(RMTS) is an online mobile cloud facility from where end users could request a mobile phone to test their mobile applications. Obviously, this does not have any emulation but some sort of remote component is there in making use of a remote real phone for its real hardware by transferring the needed application over the network onto that real phone on which the application has to be installed and run.[23]

Cloud Computing- Banking on the Cloud just specifies the potential Mobile Virtualization techniques available to be deployed on Cloud for the apparent advantage of Cloud computing. But again, these techniques are just for accessing the existing real hardware(including sensors) within the guest hosted by a guest on a real smartphone.[24]

Enterprise 2020: Examining VMware's View of the Road Ahead for the Knowledge Worker focuses on VMware Mobile Virtualization topic as already mentioned.[25]

Cooperative solutions for Bring Your Own Device (BYOD) talks about enterprise solutions for secure workplace environments based on mobile separation techniques, for example, VMware Horizon Mobile.[26]

Xen on ARM: System Virtualization using Xen Hypervisor for ARM-based Secure Mobile Phones talks about a design of system virtualization for ARM CPU architecture and describe implementation of prototype called Xen on Arm using Xen hypervisor. Again, this allows virtualized access to the underlying real hardware(including sensors) as already pointed out above in other works.[27]

13.2.1 Conclusion

A lot more including [28], [29], [30], [31], [32] form an endless list of works on Mobile Virtualization that are virtually impossible to put in a finite list here. And they all talk about virtualizing the access to the underlying hardware(including sensors) for the sake of the guest operating systems running on a real smartphone. But, the sole concept of remoteness and emulation of sensors are absent in these works but the current work.

13.3 Works focusing on remote and real-time sensor data but not emulation of sensors

All the below mentioned works are related to the current work of Sensor Emulation, in **limited** senses as none of them emphasize the emulation of sensors remotely and in real-time which is the main concept of the current work. And the benefit of emulation of sensors remotely in real-time have been discussed in depth in the Abstract, Introduction, and Use Cases sections of this document.

PRISM: Platform for Remote Sensing using Smartphones comes closer to the current work in terms of the remote-sensing features having been accomplished. But, it doesn't emulate sensors at all. It talks about an distributed network infrastructure built for the remote-sensing using the smartphones' real hardware sensors. So, the relation with the current work ends with the remote-sensing.[33]

SociableSense: Exploring the Trade-offs of Adaptive Sampling and Computation Offloading for Social Sensing shares a common factor of real-time sensing based on the adaptive sampling of the real hardware sensors but the relation ends there - no emulation of sensors at all.[34]

The Mobile Sensing Platform: An Embedded System for Activity Recognition is about a small wearable device designed for embedded activity recognition with the aim of broadly supporting context-aware ubiquitous computing applications. But apart from the common factor of usage of sensors of a mobile in a context-sensitive manner which is in a way considering the real-time nature of the surrounding environment, there's no other relation with the current work, especially in terms of emulating the sensors of a smartphone.[35]

Location-log: Bringing Online Shopping Benefits to the Physical World with Magnetic-based Proximity Detection is a mobile phone and cloud based system that brings the benefits of online shopping to the physical world. It uses magnetic-based proximity detection technology to obtain the physical proximity relationships between customers and shops in a reliable and convenient manner. But the relation ends with the common factor of using the magnetic-sensors in real-time with an external dongle attached to the smartphone and has no concept of emulation of sensors in it.[36]

Design and Evaluation of a Wireless Magnetic-based Proximity Detection Platform for Indoor Applications quantifies the notion of "within arm's reach" using "proximity zone", and propose a methodology that empirically and systematically compare the proximity zones created by various wireless technologies. It talks about a wireless proximity detection platform based on magnetic induction - PULSAR. But, the common factors end with the wireless-ness and sensor-usage in the system being designed and evaluated. And it has nothing related to emulation of sensors.[37]

Demo: Creating Interactive Virtual Zones in Physical Space with Magnetic-Induction uses real hardware sensors to sense the magnetic induction for creating virtual zones for real-world objects. Apart from the usage of sensors in real-time to facilitate the interactive virtual zones, there is no concept of emulation of sensors happening and hence the relation ends there.[38]

EmotionSense: a mobile phones based adaptive platform for experimental social psychology research is about a mobile sensing platform for social psychology studies based on mobile phones. It discusses about the ability of sensing individual emotions as well as activities, verbal and proximity interactions among members of social groups. But the common factor ends with that. No emulation of sensors are being dealt with.[39]

Mobile Sensing for Mass-Scale Behavioural Intervention discusses about the need for scalable sensing systems that can fully exploit the sensing abilities of smartphone to a large extent. It also discusses about the need for no manual intervention in such a scaleable system intended to be automatically configurable and updateable and so on. But, it's only an abstract discussion and not a core implementation oriented presentation of possible issues and challenges that would come in the way of building such a Mobile Sensing system and has no insight into the emulation of sensing system which is what the current work is all about.[40]

The science of behaviour change just talks about potential usage of mobile sensors in a wide range of use case scenarios emphasizing health of an individual. But, has no more relation with the current work.[41]

Participatory Sensing talks about everyday mobile devices, such as cellular phones, to

form interactive, participatory sensor networks that enable public and professional users to gather, analyze and share local knowledge. That's it and no more common factors with the current work other than using the network of real hardware sensors of a real smartphones for the sake of Participatory Sensing.[42]

Darwin Phones: the Evolution of Sensing and Inference on Mobile Phones is about Darwin that advances mobile phone sensing through the deployment of efficient but sophisticated machine learning techniques specifically designed to run directly on sensor-enabled mobile phones (i.e., smartphones). But the relation in using the Mobile sensors ends there. No emulation of sensors are being dealt with at all.[43]

Feasibility of Mobile Phone-Based Management of Chronic Illness cleanly outlines the benefits of Mobile Phone Sensors based monitoring of patients' health concerning the examination of their possible chronic illness. But it doesn't really talk about how to achieve such an effective system.[44]

Opportunities and Challenges for Smartphone Applications in Supporting Health Behavior Change: Qualitative Study is one more study that discusses the benefits of remote sensing using Mobile Phones. Just that it's only a study and not a presentation of accomplishing such a work.[45]

vNurse: Using virtualisation on mobile phones for remote health monitoring comes very closer to the current work in providing the sensor readings reportedly in a remote manner in real-time. However, the paper though claims that the sensor readings can be made available in real-time, its evaluation section **doesn't** mention any such evaluations about the **time taken** for the real hardware sensor readings to reach from the real smartphone to the remote-monitoring systems over some network infrastructure like IPv4, etc. It is also not providing any sensor emulation which is the sole concept of the current work which has many benefits discussed in the beginning of this document. Moreover, it involves huge implementation effort when compared to the current work which is as **simple** as only about inserting socket-communication logic in the Sensor HAL modules to exchange the real hardware sensor readings in a seamless manner - a generic lightweight solution. Whereas, vNurse involves a whole lot of relatively **complex** entities in order to accomplish its task that calls for a huge overhead when compared to that of the current work. First of all, it has guest OS image mounted as a virtual file system on top of Android 2.2 host OS on a HTC smartphone. Second of all, the real hardware sensor readings are being transmitted over the network to the remote-monitoring systems from this Guest OS and **not** from the host OS. Moreover, the guest and host(note that this is entirely within the real smartphone itself whereas the current work has no such guest running in it but only the default host - any Android release) interact through an internal network connectivity. In addition, there is a Python daemon that's being used to retrieve the underlying real hardware sensor readings using the Android APIs. Now, it's very apparent to say that this Python daemon running

at the application layer will be inefficient in terms of time taken to retrieve the underlying real hardware sensors' readings when compared to that of the current work which has its Sensor Emulation logic implemented way below the Application Layer and closer to the metal(the hardware including sensors, with only Linux kernel between them). As if this is not enough, there is one more Python daemon within the Guest OS which is actually responsible for sending out the sensor readings obtained from the host. Now, though time taken for the sensor readings in reaching the remote-monitoring systems have not been discussed at all, one can easily understand that the efficiency(being closer to the metal) and simplicity(less overhead and generic in terms of Sensor Emulation at HAL level as opposed to Application Layer level with no additional work for the applications in using the sensor readings on the emulator) of the current work outsmarts the relatively complex and inefficient(in terms of time taken) vNurse. And lastly, one of the requirements of the current work is that the sensor-based applications deployed in an emulated smartphone environment must be able to utilize the real hardware sensor readings obtained remotely in real-time, with absolutely no modification. This very requirement having accomplished by the current work successfully makes it generic, portable, and demand-less as discussed in the challenges chapter earlier. All of these are not addressed by vNurse may be because it's not meant for such a purpose but when comparing the current work with vNurse, these are the differences and advantages of the current work over vNurse.[46]

HARMONI: Context-aware Filtering of Sensor Data for Continuous Remote Health Monitoring implements a Middleware(which usually sits above HAL and below Application Layer in a standard software architecture) and hence in the first place is less efficient in terms of time taken when compared to the current work based on HAL being more closer to the metal. Next comes the absence of real-time transmission of sensor readings(which is so for a reason as the paper claims that its for making meaningful comparisons because uncontrollable physiological and environmental variations make it impossible to get the exact same data stream from two different sessions). However, real-time can't be compromised as it's one of the strengths of the current work and hence HARMONI, lacking it isn't effective when real-time sensor data is needed. Obviously, it's not about Sensor Emulation and hence the chance to run sensor-based applications using those sensor readings in an emulated smartphone environment is ruled out.[47]

13.3.1 Conclusion

There are lot more similar healthcare related works including [48],[49],[50],[51] which emphasize on only remoteness and real-time aspects of sensor data being transmitted but they don't really talk about the emulation of sensors which the current work does. Thus, the current work on Sensor Emulation is unique.

13.4 Works focusing on wireless sensor networks but not on sensor emulation

Below are the works that are solely about accomplishing a network of sensors or emulating the wireless sensor networks. These are being listed and discussed in short since they involve network-based exchange of sensor readings from one node(may be PDA, etc) to another either in real-time or in a record-replay fashion. However, these anyway lack the Sensor Emulation in a virtualized smartphone environment which is the heart and sole of the current work of Sensor Emulation.

Radiation Detection with Distributed Sensor Networks focuses on using Distributed Sensor Networks to detect harmful radiation level. It uses PDAs to process sensor data from radiation detectors to which they are attached and these PDAs relay the sensor data to other nodes. It also talks about Sensor Simulation Software but nothing about Sensor Emulation which is more realistic which is achieved by the current work in real-time remotely.[52]

Distributed Sensor Networks for Detection of Mobile Radioactive Sources focuses on examining the distributed sensing problem by modeling a network of scintillation detectors measuring a Cesium-137 source. It examines signal-to-noise behavior that arises in the simple combination of data from networked radiation sensors. No emulation of sensors but only a simulation model was developed for this.[53]

Wireless integrated network sensors(WINS) is about deploying many real hardware sensors in a wireless network mesh model and let the associated systems communicate and propagate the sensor readings to a centralized system to perform the needed action. But, as said, there is no emulation of sensors happening in these systems which is crucial to the current work as its one of its strengths.[54]

Wireless Integrated Network Sensors: Low Power Systems on a Chip, similar to the above uses WINS but again no concept of emulation of sensors.[55]

Self-Organizing Sensor Networks for Integrated Target Surveillance is about a novel self-organization protocol and describe other relevant, indigenous building blocks that can be combined to build integrated surveillance applications for self-organized sensor networks. It has been experimented in both simulated and real-world platforms. But again, it's not based on Sensor Emulation but based on real hardware sensors connected via wireless distributed network and the current work embraces emulation of sensors, thus being different and unique.[56]

Air-dropped sensor network for real-time high-fidelity volcano monitoring has some relation to the current work since it involves real-time sensor data transmission

between real sensors deployed in a wireless sensor nodes network.[57][58][59]

Other similar works like [60],[61], a lot more that can be listed here deal with the monitoring the natural events using wireless sensor networks as said earlier but with no Sensor Emulation in them being accomplished.

13.4.1 Conclusion

There are a lot more works like the above on Wireless Sensor Networks which employ many real hardware sensors connected through distribute network to coordinate with each other and help in successfully monitoring remote locations possibly in real-time. They also usually involve simulation models to test their feasibility in real world. However, these works don't focus on emulating sensors in a virtualized environment remotely in real-time which is what the current work is all about and emulation of sensors has many benefits which have been discussed in Abstract, Motivation and Use Cases chapter in detail. Thus, the current work of Sensor Emulation is unique.

13.5 Miscellaneous

Medusa: A Programming Framework for Crowd-Sensing Applications is about crowd-sensing which is nothing but a capability that harnesses the power of crowds to collect sensor data from a large number of mobile phone users. Apart from aggregating the sensor data from the real hardware sensors of real phones obviously not in real-time though over the network(and could be said as remote in some way), this work has no other relation with the current work.[62]

Open data kit sensors: a sensor integration framework for android at the application-level focuses on driving an external sensor from a real phone over USB or Bluetooth as of now. However, it doesn't support WiFi based driving as of now, so no remoteness supported unlike in the current work. It doesn't talk about emulating sensors in a virtualized platform but only a real phone unlike in the current work. It ***must*** be noted that emulating a standard Android sensor in a real phone (in addition to the virtual phones) is also one of the use cases(as an out of box one!) of the current work as listed in the Chapter 2 of this document. This work has an abstraction framework to access both internal sensors of a real phone and the external sensors but the framework being at application-level obviously and definitely is not as efficient as the current work's HAL-based Sensor Emulation which is much closer to the metal. Moreover, there is a usage of interfacing board between the external sensor and the real phone and there is a sensor driver for driving the external sensors from the real phone via the interfacing board - all of these are absent in the current work as the current work doesn't focus on driving the external sensors at all or in other words, the current work focuses on emulating real hardware sensors present in a real Android phone

itself, and hence no question of comparison on these lines. As a direct consequence of this common abstraction framework and hence the common user-space sensor drivers for accessing both built-in and external sensors, the sensor-based applications needing to access even just the built-in sensors, have to apparently face the overhead(may be little but it's anyway absent in the current work) of passing through this extra layer of the abstraction framework before reaching out the existing standard Android layers to get the needed built-in sensor reading. Though this work claims to reduce the application's lines of code in accessing even the built-in sensor(accelerometer, as mentioned in its paper's table 8), the lines of code are just shifted towards the sensor driver. True that the sensor driver will be written once for any sensor(internal or external) and any number of applications can use the same driver every time, but with respect to the current work which focuses on internal sensors, the concept of the user-space sensor drivers is totally absent and hence the corresponding number of lines of code and thus, the total number of lines of code will be still lesser in the case of current work. It must be noted that these sensor drivers are implemented in user-space and in Java, and hence the obvious performance impact as opposed to kernel-space drivers closer to the metal implemented in C. So, lastly, because of this performance impact, this work sacrifices the real-time aspect in getting the sensor data to ***some extent*** whereas the current work doesn't compromise on the real-time aspect of the Sensor Emulation at all since there are no such extra level of implementation like these sensor drivers in user-space even for accessing underlying real hardware sensors if not the external ones. All in all, this work comes ***very close*** to the current work as it emulates external sensors in a real phone in a wireless (but not remote) manner almost in real-time, but there ends the relation as it doesn't talk about emulating sensors in a virtualized platform but only on real phones.[63]

Enabling Multiple BSN Applications Using the SPINE Framework focuses on using the SPINE framework to support heterogeneous health-care applications without redeployment of the code running on the nodes. It's actually about Wireless Body Sensor Network based on SPINE where real hardware sensors are networked together by some wireless means. The relation with the current work is that it emulates the sensors in a virtual platform(but no yet in Android as reported) but these emulated sensors are to be fed with recorded sensor data from real hardware sensors and hence no real-time aspect is applicable for these emulated sensor readings. Thus the relation is limited with no real-timeness in emulation and also not being available for Android as of now.[64]

[65],[66],[67],[68],[69],[70],[71], and [72] are about Wireless Sensor Networks or Wireless Body Sensor Networks and have no concept of emulation of sensors remotely in real-time.

13.5.1 Conclusion

There are a lot more works than those can be listed here, that

just have real sensors' usage in common with the current work but with no Sensor Emulation happening. So, it's virtually impossible to list all of them individually as they are endless in numbers if even minute relations are to be considered under this Chapter of related works.

13.6 Conclusion

In short, there is no such Sensor Emulation work which is system level generic work and emulates the ***ten sensors*** the current work emulates as described throughout this document, laying emphasis on ***remoteness*** and ***real-time*** aspects.

Chapter 14
Conclusion and Future Work

With this work on Sensor Emulation, any sensor-based Android application can be run on the virtualized Android-x86 running on any x86 emulator like Qemu as long as the application is compatible with the emulator, especially in terms of graphics. This opens up a whole new opportunity for the Android developers to test their sensor-based applications in an emulated environment with the real device sensor readings, which was previously impossible and it demanded to have real Android devices installed with the necessary Android OS release, and so on. Due to the real-time nature of the sensor readings available on the emulated Android-x86, there is absolutely no difference in running the same applications on a real Android device and on the virtual Android device. Hence, the sensor-based applications can be accurately tested in a virtualized Android-x86 environment.

14.1 Testing with the standard Android emulator

The current work described in the paper is about pairing a real Android device with virtualized Android-x86. But, the standard ARM based Google's Android emulator could also be used with no extra work by the simple logic that Android-x86 is more or less the default main Android code with the target being x86 platform. However, the current work has not been tested. So, it's one of the interesting future works. Also, since the standard Android emulator now includes an Intel x86 Atom image as well, testing the current work with that as well would be interesting to work on going forward.

14.2 One-server-many-clients

One-server-many-clients could be implemented to take this Sensor Emulation to another level enabling the sensor-based testing experience in an emulated Android-x86 to a whole new dimension. This would enable pairing a single real Android device with many virtual Android devices. In addition, any other sensors not already being emulated could be added to the to-do list of Sensor Emulation.

14.3 Two-way communication : one-real-many-but-one-master-virtual

As of now, the communication between the real and virtual phones is only one way wherein the real phone has the full control over its real hardware sensors and communicates their behavior to all the connected(paired) virtual phones for their emulated sensors. Enabling two-way communication between the real and virtual phones, wherein one real phone feeds many virtual phones wherein only one of them is a master virtual phone will be an interesting future work as the master virtual phone can also appropriately control the real phone like initiating calibration action on its emulated sensor so the paired real phone will calibrate the specific real hardware sensor get, etc and all the virtual phones including the master virtual phone can see this calibration in their specific emulated sensors. Thus two-way communication is enabled between real and virtual phones.

14.4 Remaining sensors

Sensors which are not being emulated as of now as mentioned in Chapter 6, could be taken up for the emulation as per the need.

14.5 Continued testing with graphics-intensive 3D games

The rigorous testing can be continued when the graphics-intensive 3D sensor-based applications get their graphical dependency issues resolved in a reliable way. For the list of graphics issues that prevented most of the 3D sensor-based games from launching itself or running after launching, one may refer the earlier chapter on “Challenges”.

14.5 Power consumption analysis

The Sensor Emulation work hasn’t been analyzed for the power consumption on the real phone though everything is asynchronously driven and is highly optimized end-to-end. So, analyzing the power consumption and comparing the results from with and without Sensor Emulation scenarios would be a potential future work.

14.6 Memory footprint analysis

The Sensor Emulation work hasn’t been analyzed for its memory footprint though no big chunk of memories are being used and the code has been free from memory leaks at the best of one’s knowledge. So, analyzing the memory footprint of the Sensor Emulation end-to-end can be a good future work.

14.6 Runtime Overhead Analysis

The Sensor Emulation work hasn't been analyzed for its overhead induced for the real phone with respect to the sensor-based applications running on it though it will be very minimal as code that has been added to Sensors HAL module is carefully optimized end-to-end. So, analyzing the runtime overhead which would anyway be minimal, will be an interesting future work.

14.7 Emulating an External Sensor

The Sensor Emulation work is limited to emulating the real hardware sensors that are standard Android ones and hence are built into the real phones. But, emulating a totally isolated sensor not defined by Android Sensors subsystem can be a challenging future work if anytime found appropriate.

Chapter 15
Bibliography

1. android-x86.org from where the Android-x86 source was obtained.
2. https://android.googlesource.com for the Samsung Crespo device specific sensor code and the virtual sensors code.
3. Online discussion groups: android-x86, adt-dev, android-contrib, android-developers, stackoverflow.com.
4. stackoverflow.com Android sensor related questions and answers.
5. Android Open Source Code online documentation in general and on HAL and sensors HAL, in specific.
6. In depth discussions with Duke University PhD student, Songchun Fan, mentioned in the header of this document.
7. Several Android applications reviews websites to determine what are the best available sensor based Android applications for testing the current work.
8. http://developer.samsung.com/forum/ for Samsung Sensor Simulator.
9. https://code.google.com/p/openintents/wiki/SensorSimulator for Sensor Simulator by Openintents.
10. http://tools.android.com/recent/sensoremulation for Sensor Emulation in Google Android tools.
11. http://developer.samsung.com/android/tools-sdks/Samsung-Sensor-Simulator for Samsung Sensor Simulator from Samsung.
12. http://developer.samsung.com/android/tools-sdks/Samsung-GALAXY-Tab-Emulator for Samsung GALAXY Tab Emulator.
13. https://developer.android.com/sdk for the Standard Android Emulator.
14. https://www.manymo.com/ for Manymo, claimed as better emulator for Android.
15. http://www.genymotion.com/ for Genymotion, claimed as a faster/the fastest Android emulator.
16. http://bluestacks.com/ for Bluestacks software to run Android games on PC.
17. http://www.xda-developers.com/android/jar-of-beans-a-portable-android-emulator/ for Jar of Beans, an Android-x86 emualtor that's discontinued.
18. YouWave for Android - Android on Windows PC for YouWave Android emulator.
19. Cells: A Virtual Mobile Smartphone Architecture. Jeremy Andrus, Christoffer Dall, Alexander Van't Hof, Oren Laadan, Jason Nieh. Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP 2011), October 2011
20. The Design, Implementation, and Evaluation of Cells: A Virtual Smartphone Architecture. Christoffer Dall, Jeremy Andrus, Alexander Van't Hof, Oren Laadan, Jason Nieh. ACM Transactions on Computer Systems (TOCS), Volume 30, Issue 3, August 2012.
21. Barr, Ken, et al. "The vmware mobile virtualization platform: is that a hypervisor

in your pocket?." *ACM SIGOPS Operating Systems Review* 44.4 (2010): 124-135.
22. Kalim, Asra. "Clouds on the Academic Horizon."
23. Huang, J-F., and Y-Z. Gong. "Remote mobile test system: a mobile phone cloud for application testing." *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*. IEEE, 2012.
24. Goel, Pragati. "Cloud Computing-Banking on the Cloud."
25. Edwards, Richard. "Enterprise 2020: Examining VMware's View of the Road Ahead for the Knowledge Worker."
26. Jaramillo, D., et al. "Cooperative solutions for Bring Your Own Device (BYOD)." *IBM Journal of Research and Development* 57.6 (2013): 5-1.
27. Hwang, Joo-Young, et al. "Xen on ARM: System virtualization using Xen hypervisor for ARM-based secure mobile phones." *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*. IEEE, 2008.
28. Brakensiek, Jörg, et al. "Virtualization as an enabler for security in mobile devices." *Proceedings of the 1st workshop on Isolation and integration in embedded systems*. ACM, 2008.
29. Mijat, Roberto, and Andy Nightingale. "Virtualization is coming to a platform near you." *ARM White Paper* (2011).
30. Gudeth, Kevin, et al. "Delivering secure applications on commercial mobile devices: the case for bare metal hypervisors." *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*. ACM, 2011.
31. Inoue, Hiroaki, et al. "VIRTUS: a new processor virtualization architecture for security-oriented next-generation mobile terminals." *Proceedings of the 43rd annual Design Automation Conference*. ACM, 2006.
32. Ryu, Euiyoul, et al. "MyAV: An all-round virtual machine monitor for mobile environments." *Industrial Informatics (INDIN), 2010 8th IEEE International Conference on*. IEEE, 2010.
33. Das, Tathagata, et al. "PRISM: platform for remote sensing using smartphones." *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010.
34. Rachuri, Kiran K., et al. "Sociablesense: exploring the trade-offs of adaptive sampling and computation offloading for social sensing." *Proceedings of the 17th annual international conference on Mobile computing and networking*. ACM, 2011.
35. Choudhury, Tanzeem, et al. "The mobile sensing platform: An embedded activity recognition system." *Pervasive Computing, IEEE* 7.2 (2008): 32-41.
36. Zhang, Ben, et al. "Location-log: Bringing Online Shopping Benefits to the Physical World with Magnetic-based Proximity Detection." (2012).
37. Jiang, Xiaofan, et al. "Design and evaluation of a wireless magnetic-based proximity detection platform for indoor applications." *Proceedings of the 11th international conference on Information Processing in Sensor Networks*. ACM, 2012.
38. Jiang, Xiaofan, et al. "Demo: Creating interactive virtual zones in physical space with magnetic-induction." *Proceedings of the 9th ACM Conference on Embedded*

Networked Sensor Systems. ACM, 2011.
39. Rachuri, Kiran K., et al. "EmotionSense: a mobile phones based adaptive platform for experimental social psychology research." *Proceedings of the 12th ACM international conference on Ubiquitous computing*. ACM, 2010.
40. Mascolo, Cecilia, Mirco Musolesi, and Peter J. Rentfrow. "Mobile sensing for mass-scale behavioural intervention." *NSF Workshop on Pervasive Computing at Scale (PeCS)*. 2011.
41. Perry, Chris. "The science of behaviour change."
42. Burke, Jeffrey A., et al. "Participatory sensing." (2006).
43. Miluzzo, Emiliano, et al. "Darwin phones: the evolution of sensing and inference on mobile phones." *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010.
44. Smith, Joshua C., and Bruce R. Schatz. "Feasibility of mobile phone-based management of chronic illness." *AMIA Annual Symposium Proceedings*. Vol. 2010. American Medical Informatics Association, 2010.
45. Dennison, Laura, et al. "Opportunities and Challenges for Smartphone Applications in Supporting Health Behavior Change: Qualitative Study." *Journal of medical Internet research* 15.4 (2013).
46. Rehunathan, Devan, et al. "vNurse: Using virtualisation on mobile phones for remote health monitoring." *e-Health Networking Applications and Services (Healthcom), 2011 13th IEEE International Conference on*. IEEE, 2011.
47. Misra, A., M. Ebling, and W. Jerome. "Harmoni: Context-aware filtering of sensor data for continuous remote health monitoring." *Pervasive Computing and Communications, 2008. PerCom 2008. Sixth Annual IEEE International Conference on*. IEEE, 2008.
48. Chowdhury, Atanu Roy, Benjamin Falchuk, and Archan Misra. "Medially: A provenance-aware remote health monitoring middleware." *Pervasive Computing and Communications (PerCom), 2010 IEEE International Conference on*. IEEE, 2010.
49. Moulton, Bruce, Zenon Chaczko, and Mark Karatovic. "Data Fusion and Aggregation Methods for Pre-Processing Ambulatory Monitoring and Remote Sensor Data for Upload to Personal Electronic Health Records." *JDCTA* 3.4 (2009): 120-127.
50. Wyne, Mudasser F., et al. "Remote patient monitoring using GSM and GPS technologies." *Journal of computing sciences in colleges* 24.4 (2009): 189-195.
51. Sha, Kewei, et al. "SPA: a smart phone assisted chronic illness self-management system with participatory sensing." *Proceedings of the 2nd International Workshop on Systems and Networking Support for Health Care and Assisted Living Environments*. ACM, 2008.
52. Mielke, Angela M., et al. "Radiation detection with distributed sensor networks." *Defense and Security*. International Society for Optics and Photonics, 2005.
53. Nemzek, Robert J., et al. "Distributed sensor networks for detection of mobile radioactive sources." *Nuclear Science, IEEE Transactions on* 51.4 (2004): 1693-1700.
54. Pottie, Gregory J., and William J. Kaiser. "Wireless integrated network sensors."

Communications of the ACM 43.5 (2000): 51-58.
55. Asada, G., et al. "Wireless integrated network sensors: Low power systems on a chip." *Solid-State Circuits Conference, 1998. ESSCIRC'98. Proceedings of the 24th European*. IEEE, 1998.
56. Agre, Jonathan R., et al. "Development platform for self-organizing wireless sensor networks." *AeroSense'99*. International Society for Optics and Photonics, 1999.
57. Song, Wen-Zhan, et al. "Air-dropped sensor network for real-time high-fidelity volcano monitoring." *Proceedings of the 7th international conference on Mobile systems, applications, and services*. ACM, 2009.
58. Song, Wen-Zhan, et al. "Design and deployment of sensor network for real-time high-fidelity volcano monitoring." *Parallel and Distributed Systems, IEEE Transactions on* 21.11 (2010): 1658-1674.
59. Huang, Renjie, et al. "Real-world sensor network for long-term volcano monitoring: design and findings." *Parallel and Distributed Systems, IEEE Transactions on* 23.2 (2012): 321-329.
60. Tan, Rui, et al. "Quality-driven volcanic earthquake detection using wireless sensor networks." *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st*. IEEE, 2010.
61. Wang, Fei, and Hongyong Yuan. "Challenges of the Sensor Web for disaster management." *International Journal of Digital Earth* 3.3 (2010): 260-279.
62. Ra, Moo-Ryong, et al. "Medusa: A programming framework for crowd-sensing applications." *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM, 2012.
63. Brunette, Waylon, et al. "Open data kit sensors: a sensor integration framework for android at the application-level." *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM, 2012.
64. Gravina, Raffaele, et al. "Enabling multiple BSN applications using the SPINE framework." *Body Sensor Networks (BSN), 2010 International Conference on*. IEEE, 2010.
65. Gravina, Raffaele, et al. "Development of body sensor network applications using SPINE." *Systems, Man and Cybernetics, 2008. SMC 2008. IEEE International Conference on*. IEEE, 2008.
66. Bellifemine, Fabio, et al. "SPINE: a domain-specific framework for rapid prototyping of WBSN applications." *Software: Practice and Experience* 41.3 (2011): 237-265.
67. Fortino, Giancarlo, et al. "SPINE2: developing BSN applications on heterogeneous sensor nodes." *Industrial Embedded Systems, 2009. SIES'09. IEEE International Symposium on*. IEEE, 2009.
68. Kuryloski, Philip, et al. "DexterNet: An open platform for heterogeneous body sensor networks and its applications." *Wearable and Implantable Body Sensor Networks, 2009. BSN 2009. Sixth International Workshop on*. IEEE, 2009.
69. Iyengar, Sameer, et al. "A framework for creating healthcare monitoring

applications using wireless body sensor networks." *Proceedings of the ICST 3rd international conference on Body area networks*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.
70. Aiello, Francesco, et al. "An agent-based signal processing in-node environment for real-time human activity monitoring based on wireless body sensor networks." *Engineering Applications of Artificial Intelligence* 24.7 (2011): 1147-1161.
71. Aiello, Francesco, et al. "A java-based agent platform for programming wireless sensor networks." *The Computer Journal* 54.3 (2011): 439-454.
72. Latré, Benoît, et al. "A survey on wireless body area networks." *Wireless Networks* 17.1 (2011): 1-18.
73. Operating System Usage Trend September – December 2013: Smartphone Operating Systems Are Driving Growth!
74. Android, the world's most popular mobile platform