2016

Miniatured Inertial Motion and Position Tracking and Visualization Systems Using Android Wear Platform

Dhruvkumar Navinchandra Patel
Wright State University

MINIATURED INERTIAL MOTION AND POSITION TRACKING AND VISUALIZATION SYSTEMS USING ANDROID WEAR PLATFORM

A thesis submitted in partial fulfillment

of the requirement for the degree of

Master of Science

By

DHRUVKUMAR NAVINCHANDRA PATEL
B.E., Gujarat Technological University, 2014

2016

Wright State University

WRIGHT STATE UNIVERSITY

GRADUATE SCHOOL

January 6th, 2017

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Dhruvkumar Navinchandra Patel ENTITLED Miniatured Inertial Motion and Position Tracking and Visualization Systems Using Android Wear Platform BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science.

Yong Pei, Ph.D.
Thesis Director

Mateen M. Rizki, Ph.D.
Chair, Department of Computer Science and Engineering

Committee on Final Examination

Yong Pei, Ph.D.

Mateen M. Rizki, Ph.D.

Paul Bender, Ph.D.

Robert E.W. Fyffe, Ph.D.
Vice President for Research and Dean of the Graduate School

ABSTRACT

Patel, Dhruvkumar Navinchandra. M.S. Department of Computer Science and Engineering, Wright State University, 2016. Miniatured Inertial Motion and Position Tracking and Visualization Systems Using Android Wear Platform.

In this thesis, we have designed and developed a motion tracking and visualization system using the latest motion tracking sensory technologies. It is one of the enabling technologies for our novel visual-inertial odometer and human anatomy based 3D Locating, Mapping and Navigation system for endoscopy and drug delivery capsules used inside GI tract. In particular, we have: i) designed and completed a cloud-based sensory data collecting, processing and storage system to provide the reliable computing and storage platform; ii) explored different data processing methods to obtain improved-quality motion results from extremely noisy raw data, e.g., by using a low pass and high pass filter, and Kalman filters; iii) developed low-complexity algorithms to support real-time data analysis; and, iv) provided real-time 3 dimensional visualizations by a Unity 3D based visualizer.

In this thesis, we have also showcased the use of application processors, which are widely used in smartphones and tablets, to develop a potentially low-cost sensor system and networks with enhanced computing, storage and networking capabilities. Specifically, we have explored Android/Android Wear, Google's open source mobile OS's, enabled smart devices, such as Sony Smartwatch 3, and their built-in sensory capabilities to build our sensory system. The completeness and maturity of such a widely used mobile platform ensure a fast prototype design and development process, as well as significantly better reliability and usability. The reliability of our sensory system is further improved through the use of a database approach, such as transactions, for the data flow from sensors to mobile platform and eventually to the cloud. Thus, our prototype design provides a working model to collect sensor data from Android Wear, and then transfer and store them into the cloud for further processing and disseminations.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my sincere gratitude to my thesis advisor Dr. Yong Pei, for guiding me through this endeavor and being a source of constant inspiration. Despite battling with thoroughly preoccupied and strenuously tight schedules, you’ve spared time to render every possible effort to help me go through this thesis work, and provided infrastructure and resources as well. Also I would like to thank my lab mates for their motivations.

Chapter 1

1. Introduction

The motivation of this research is to demonstrate the latest capabilities in motion capture and analysis and their applications in Healthcare domain to help people live healthier lives. It has been reported that nowadays 40% of the people suffer from Lower Back Pain [1]. For instance, for those people who sit almost the whole day in front of the computers, research shows that almost everyone (up to 85-90%) among them has suffered from lower back pain at some point in his/her life [1]. One of our goal is to develop technologies to help identify the cause of lower back pain and assist the corresponding therapy process. Another potential application of this research is to enable automatic locating and mapping capabilities inside GI tract system in order to provide the high-precision navigation capabilities for endoscopy and drug delivery capsules used in GI medicine. We expect that these proposed new technologies will provide improved workflow, accuracy and efficiency in sports, fitness and healthcare fields.

1.1. Overview of Motion Technology

Motion capture devices have been widely used to measure motion and movement of an object in various fields, such as entertainment, sports, military and health care applications [3]. In this thesis, we are primarily interested in developing position and motion tracking technologies in the health care domain for such applications as human body joint gait analysis, locating and mapping the endoscopy and drug-delivery capsules within the GI tract, and etc. As a result, we are particularly interested in designing non-optical and miniaturized motion tracking systems, such as inertial motion tracking capabilities as evident in motion sensors used in most smartphones, such as the *3x3x1 mm* 9-axis motion sensors by InvenSense as shown in Fig.1.1. [2].

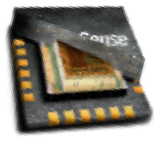


Figure 1. 1 3x3x1 mm InvenSense's MPU-925x Motion tracking unit (3-axis gyroscope, 3-axis accelerometer and 3-axis compass) [2]

The combination of Accelerometer, Gyroscope and Magnetometer in a single Inertial Measurement Unit (IMU) has been commonly used in most of today's motion tracking sensors [3]. Such highly miniaturized sensors are made possible by MEMS (microelectromechanical systems) technologies, as shown in related products by InvenSense, Qualcomm, EM Micro and other vendors. Such motion units enable the motion tracking capabilities on the smartphone and wearable technologies, and Virtual and Augmented Reality (VR/AR) technologies [3].

As evident in such popular devices as Apple Watch and Microsoft Band, such inertial sensors can be used for capturing accelerations and orientation data of an objects and then, through sensor data fusion algorithms and applications, to produce motion analysis in velocity and position, leading to meaningful applications in different areas. Moreover, wearable devices with embedded IMUs can enable clinical specialists to track, manage and train the patient on a certain movement. It will also help in sports where athletes can study movement, and then refine and improve their techniques [3].

1.2. Overview of Android Wear and Handheld Technology

Android is an operating system based on Linux kernel designed for mobile devices such as smartphones, tablets and also for wearable devices such as android watches. As an open source technology, it provides us freely the Application Programming Interface (API) for application development. APIs provide commonly used utilities, and are made available for speeding up the development with enhanced efficiency and reliability, for broad spectrum of usage in various applications. Following are the features that attract us to use this operating system for our related prototype developments.

1. Most of the Android devices have built in sensors for motion, orientation and other environmental conditions. Thus, we can directly use these types of devices as a motion capture device. It provides a much more cost-efficient alternative to use the related development kits from the IMU vendors. This is particularly beneficial for research purpose.
2. Android provides a reliable, secure and consistent OS for development and distribution of new applications.
3. Android devices support intuitive interactions between user and application using Graphical User Interface.
4. Android device’s functionalities are written in Android software development kit (SDK), which uses Java and sometimes C/C++ programming languages that have access to the Android APIs.

5. Android platform provides Automatic Memory Management functionality and it is also easy to integrate and customize the operating system.

In short, Android Wearable and handheld devices are low in price and gives us nice features for development purpose. Wearable technology and research opens new areas of innovation, particularly when connected and attached with the Cloud's big data capabilities, to support sports, fitness, virtual reality and healthcare. It is important to note that Android/Android Wear provide fully-fledged networking capabilities to be connected through a handheld using, e.g., Bluetooth, to the Internet and Cloud. As a result, in this research we will build our prototype system of 3D human body motion tracking and visualization using Android/Android Wear devices' built-in sensor capabilities.

1.3. Cloud Based Prototype to Collect Sensor Data using Android Wear

In this section, we discussed our system that collects data from Android Wear Motion sensors. We will establish a communication channel between Android Handheld and Android Wear devices for actuation/control and data transfer. Here, we are presenting an approach for the complete data flows: i.) to send and synchronize large sensor datasets between android handheld and Android Wear; ii.) followed by storing the motion sensor data in Database on Android handheld, and then iii.) transfer that dataset into Cloud for storage and processing.

We will then process the collected sensor data to find 3-Dimensional accelerations, velocity and position using sensor filter techniques and use Unity 3D-based 'Visualizer' to visualize 3D position trajectory system. Following are the major advantages of this approach.

1. Motion sensor data is collected through the well-established Android Wear OS. Thus large dataset can be conveniently and reliably synchronized between nodes.
2. Provide Bluetooth connectivity and data transfer facility between handheld and Android Wear.
3. Provide local Database to store on handheld side when connectivity is not available; and later store in Cloud for access from anywhere.
4. Android Wear can be easily attached with human body, so we can accurately track human body motion activity with minimized interference to the person.

We have developed a prototype system to show the entire workflows. We will discuss technical details, development procedure and implementation details in later chapters of this thesis.

Chapter 2

2. Sensor System and Motion Capture Devices

In this chapter, we will discuss in detail what sensor capabilities and devices we need for our system, and why we choose Android Wear and Handheld Technologies. Particularly, we want to illustrate the viability of using mobile application processors and mobile OS for wireless sensors and network system designs.

2.1. What we need for our System

To detect the 3-dimensional human body motion and position tracking and visualization, we need high quality miniatured motion sensors, especially with high sampling rate and processing capabilities. Thus, we chose to explore the use of 9 Axis accelerometer+gyroscope+compass sensor hub, which can produce, through sensory data fusion, virtual or synthetic sensors like linear acceleration, gravity, rotation vector and orientation for various applications. These sensors are widely available today, e.g., by MEMS, InvenSense, QUALCOMM, EM Micro and others, thanks to the huge market demand in smartphones and tablets. The synthetic sensors can be developed, e.g., by AOSP (Android Open Source Project) [4].

Conventionally, for research prototype development, you can use the Development Kits from the vendor together with the sensor hub chip and Arduino boards. Arduino is an open source electronics platform for hardware and software applications. It can be used to read inputs from sensors and process and turn it into the desired output.

Programs based on Arduino programming language will be developed for data processing. However, there are serious limitations of this approach in cost, effort and complexity. Individually and collectively, the components required to build a prototype could be very costly. For example, InvenSense 9 Axis IMU development Kit alone costs over $500 [2]. Additional Arduino boards to augment the sensory system capabilities in computing, storage and networking may add up too much higher prices. Moreover, this approach requires significant developing and testing effort and time due to its primitive hardware/software co-design nature.

As a result, in this thesis, we want to illustrate the viability of using mobile application processors and mobile OS for wireless sensors and network system designs. We are looking at smartphones and wearable devices as an alternative platform, which provide all our needs for sensor capabilities, computing and networking, and provide us with a convenient development environment

For example, most Android Wearable devices contain all the required features and processing capabilities to demonstrate wireless sensors and network designs. Furthermore, Android Wearable can connect with Android handheld device to augment its capabilities in storage, processing and networking, through synchronization of a high volume of sensor data. We will also look at cloud-based solutions to further augment these capabilities, so one can store, process and retrieve a high volume of sensor data.

Next, we will discuss Android Wear sensory compatibility, processing compatibility and other advantages in the following sections, and provide arguments for Android Wear devices as a better choice over others for our prototype developments.

2.2. Android Wear as a Motion Capture Device

There are many commercially available Android Wear devices, particularly watch-like devices, with different sizes, shapes, styles and features, like, e.g., Moto 360, Samsung gear live, Sony smartwatches, etc. [18]. In our research, we will use Sony smartwatch 3, which runs Android Wear, as a motion capture device. Priced at about $150, it provides a very low-cost but highly robust, reliable and capable platform for developers as well as users [18].

2.2.1. Sony Smartwatch 3(SWR50) Specification

Sony Smartwatch 3 Android Wear is a standalone smart device with built-in processors, storage and sensory capabilities. It also provides connectivity to Android Phone or Tablet (Android Handheld) devices with version Android 4.3 or later [6]. It has maximum scratch resistance and durability, waterproof protection and a stainless-steel body. It also provides other features for Android Wear development purpose as illustrated in the following specifications in Table 2.1 [6].

Figure 2. 1 Sony Smartwatch 3 (SWR50)

Processor	1.2 GHz, Quad-core ARM® CortexTM A7
Size	36 x 10 x 51 mm
Weight	38 grams
Display	Transflective TFT LCD, multi touch Capacitive, 320x320 pixels resolution
Memory	4 GB internal storage, 512 MB RAM
Sensors	Accelerometer, Magnetometer, Gyro, Ambient light sensor
Connectivity	4.0 Bluetooth, Wi-Fi, GPS, NFC,USB
Battery	420 mAh Li-polymer, less than 1 hour charge time, standby up to 96 hours
Price	Around 150 USD
Operating System Compatibility	Android version 4.3 and later

Table 2. 1 Android Wear Teardown and Specification [6]

2.3. Why we choose Android Wear (Sony Smartwatch 3)

Following are the brief descriptions about advantages and features of Android Wear as networked sensor platforms.

2.3.1. Processing Capabilities

Sony Smartwatch 3 adopts a new wearable platform from Broadcom. It is based on the Broadcom System-on-chip (SoC) platform which contains a 1.2 GHz Quad-core ARM Cortex A7 processor with a highly integrated power management IC that assures low power consumption [6]. Ultra-low power circuits, energy efficient communication and displays become important components in making of new smartwatches. The OS - Android Wear provides efficient memory and storage management and processing of the 512MB RAM and 4GB storage, capable of managing high volume of data sufficiently for most sensory system's needs [6]. This processing capability is significantly better when compared to other platforms such as Arduino. In addition, EM Microelectronics provides the motion tracking sensors built into the SoC chip.

2.3.2. Sensors Capabilities

The device's sensory capabilities are the main reason behind using Sony smartwatch 3 as a part of this research. Here we are describing availability of each sensor, particularly on those useful for motion tracking.

2.3.2.1. Overview of Android/Android Wear Built-in Sensors

Most if not all Android Wear devices have a sensor hub chip that contains either a 9-Axis or 6-Axis MEMS (Microelectromechanical Sensors) motion sensor that is made from a silicon chip. These sensors are capable of providing raw sensor data in high rate (about 200 sample per second) and precision to use in three-dimensional motion tracking devices. The Android platform supports the following different kinds of sensors [7].

1. Motion Sensors: accelerometer, gyroscope, gravity and rotational vector sensors used to measure acceleration and rotational forces along axes.
2. Position Sensors: GPS, orientation and magnetometer sensors.
3. Environmental Sensors: barometer, photometers, thermometers sensors used to measure various environmental parameters.
4. Radio Sensors: Transceivers also provide radio signal strength at certain radio bands.

In this thesis, we will use Android-provided Sensor APIs to extract raw sensor data from these sensors. Android sensors can be classified into two different classes: i.) raw sensors, and ii.) synthetic sensors [4]. Android raw sensors abstract directly relate to one of the physical sensors available on the device, e.g., accelerometer, gyroscope and compass sensors. Synthetic sensors, on the other hand, are abstract or virtual sensors that are created by fusing data from two or more raw sensors. For example, the Linear acceleration Sensor is produced in some devices by using Accelerometer + gyroscope sensors.

Next, we will look at Google Asus Nexus 7 and Sony Smartwatch 3, as specific cases, for more details on Android sensory capabilities.

2.3.2.2. Google Asus Nexus 7 sensor specification

Google Asus Nexus 7 can be used as an Android motion capture device. This device contains a sensor chip with 9-axis accelerometer, gyro and compass sensors, as well as other raw and synthetic sensors. We have used a sensor API to list out all the available sensors in a Nexus 7 tablet [7]. As detailed in Table 2.2, we can use MPL accelerometer or linear acceleration sensor for sensor data collection before further processing to estimate velocity and position.

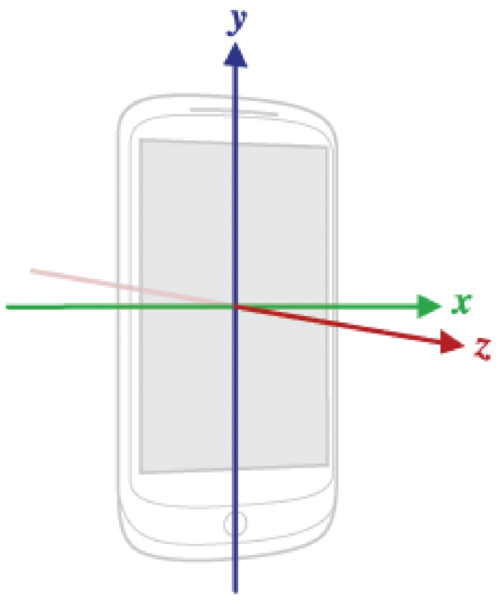
Sensor Name	**Vendor Details**	**Use of Sensor**
MPL accelerometer	InvenSense	Measure Acceleration applied to devices including gravity
AKM magnetometer	AKM	Measure magnetic field
MPL gyroscope	InvenSense	Measure the rotation around axis
Orientation	Qualcomm	Measure orientation combination of angle and axis
Light sensor	LSC	Detect current ambient light
Rotation Vector	Qualcomm	Measure orientation of device
Gravity sensor	Qualcomm	Measure direction and magnitude of gravity
Linear acceleration	Qualcomm	Measure acceleration excluding gravity

Table 2. 2 List of Available Sensors

2.3.2.3. Nexus 7 Device Coordinate System

Each motion sensor extracts a sensor data along one of its three coordinate axes. Following are details of sensor data coordinates and values for Nexus 7 [4].

1. X axis is horizontal with a positive value on right and negative on left.
2. Y axis is vertical with a positive value upwards and negative value downwards.
3. Z axis is positive values in front of the screen.



Device coordinates do not change when the device is in portrait mode or landscape mode. Figure 2.2 displays the device coordinate system in Asus Nexus 7 Tablet.

Figure 2. 2 Nexus 7 Device Coordinate System [7]

2.3.2.4. Available Sensors in Sony Smartwatch 3

We can conveniently use Android Sensor API to find out what sensors are available in Sony smartwatch 3 and its corresponding vendor. This information is obtained through sensor. getVendor () and sensor. getName () methods [7].

We will look at each motion-related sensor as has been listed in Table 2.3.

Sensor Name	**Vendor Details**	**Use of Sensor**
em8170 accelerometer	EM Micro	Measure Acceleration applied to devices including gravity
em8170 magnetometer	EM Micro	Measure magnetic field
em8170 gyroscope	EM Micro	Measure the rotation around axis
em8170 quaternion	EM Micro	Represent orientation and rotation in three dimensional
em8170 orientation	EM Micro	Measure orientation combination of angle and axis
BH1721 Light sensor	Rohm	Detect current ambient light
em8170 step counter	EM Micro	Count the number of steps by user

em8170 step detector	EM Micro	Notify when user take a step
em8170 tilt sensor	EM Micro	Measuring tilting in reference plane
Gravity sensor	AOSP	Measure direction and magnitude of gravity
Linear acceleration	AOSP	Measure acceleration excluding gravity

Table 2. 3 List of Built-in Motion Sensors in Sony smartwatch 3 Android Wear

(i) Raw sensors: Accelerometer, magnetometer and gyro are the main three raw sensors available in Sony smartwatch 3 Android Wear. Brief descriptions of these sensors are below [4].

1. Accelerometer: Sony Smartwatch 3 contains a 6-Axis Accelerometer and gyroscope IMU, so each sensor provides data over 3 axes. Accelerometer sensor measures the acceleration along three axes. The measurement is reported towards X, Y, Z fields. Accelerometer measures both gravity and physical acceleration. For example, we can use accelerometer sensor data to analyze a hand movement which is of interest in sports, health and other fields. However, Accelerometer and gyroscope sensory data processing are necessary and complicated due to its dynamic

range, wide operation frequency, high sensitivity and noisy nature.

2. Gyroscope: This sensor is used to measure rotation of the device around three axes. Rotation is positive towards counterclockwise direction which describes positive values of X, Y, Z field axes. The readings are calibrated using temperature compensation, factory scale compensation and online bias compensation.
3. Magnetometer: This is a non-wake up sensor that reports the ambient magnetic field together with a hard iron calibration estimate. It can be used to produce synthetic sensors such as orientation and rotation vector sensors with collaboration of accelerometer and gyroscope.

(ii) Synthetic sensors: linear acceleration, gravity and orientation sensors [4].

1. Linear acceleration: In Sony smartwatch 3, linear acceleration is provided by Android open source project (AOSP) which is developed using accelerometer and gyroscope (if present); otherwise, using accelerometer and magnetometer. Linear acceleration measures device acceleration force excluding gravity. In other words, you can

calculate linear acceleration by subtracting the output of gravity sensor from the output of accelerometer sensor. In this research, we will use the linear acceleration data to estimate the position trajectory which is further described in later chapter of this thesis.

2. Gravity: This sensor is used to measure direction and magnitude of gravity in the device coordinate frame. When the device is on, gravity sensor readings are the same as Accelerometer sensor readings when at rest, around 9.8 m/s^2.
3. Orientation: This sensor is deprecated and produced using Accelerometer, Gyroscope and magnetometer raw sensors. It is used to measure azimuth (angle between magnetic north direction and the Y axis), pitch (rotation around X axis, with positive values when the Z axis moves towards the y axis) and roll (rotation around Y axis, with positive values when the X axis moves towards the Z axis) around the axis.

2.3.2.5. Device Coordinate System in Sony Smartwatch 3

In Sony Smartwatch 3, the 6-axis accelerometer and gyroscope, and 3-Axis magnetometer sensor hub reports values using the device coordinate system. The device coordinate system is depending on each type of device. Sony

Smartwatch 3 measures three-dimensional sensor reading using the following coordinate system which is when the device is viewed in default orientation [4].

1. X – Axis is horizontal with positive values on right and negative on left.
2. Y – Axis is vertical with positive values on upwards and negative on downwards.
3. Z – Axis is positive values in front of the screen.

The coordinate system is fixed and not changed when the device goes from portrait to landscape mode.

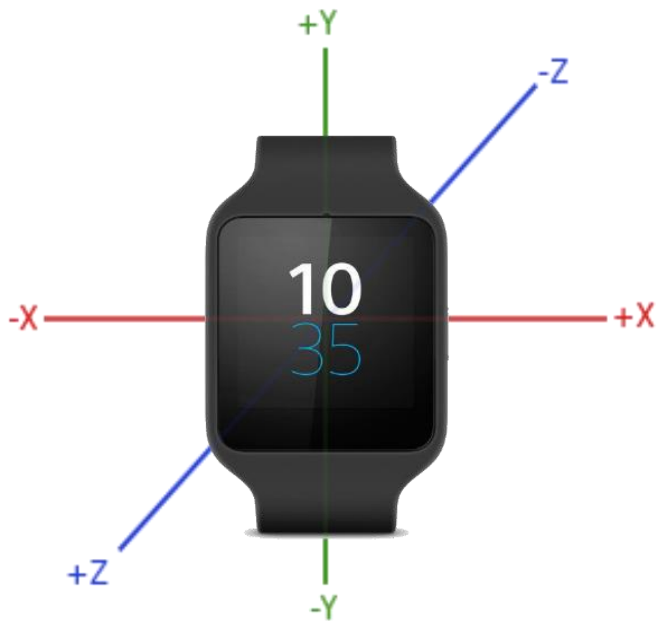


Figure 2. 3 Coordinate System in Sony Smartwatch 3

2.3.2.6. Sensor Performance: Sampling Rates & Accuracy

1. Sampling Rate: Sampling rate is the average number of samples obtained per second by the sensor [4]. Higher sampling rate implies higher temporal resolution of the sensor, but at a cost of higher power consumption. Today's MEMS sensors can provide a range of different sampling rates to fit different needs. Through the Sensor API, we can control the sampling rate by setting the inter-sample period. It also provides the corresponding Timestamp of the reading, a measurement in nanoseconds at which events occur. We can track the actual inter-sample time-interval for each measure by calculating the difference between successive timestamps. Sony smartwatch 3 supports sampling rate up to 200 samples per second, which is equivalent to 5 milliseconds in average. However, we have observed that the actual inter-sample time-interval may be varying within a small range around 5 milliseconds. As a result, it is also important to track the actual inter-sample time-interval when we process the data to estimate velocity and position information. In our research, we have pushed the envelope of the sensory capabilities of the device, and operated it at its highest sampling rate in order to sufficiently capture the high-speed motions. However, it should be noted that it is not always necessary to operate at such a high rate. We should choose the appropriate operating rate based on the application's need, and also consider the factors in battery usage,

storage and processing demand. A major challenge when operating at higher sampling rate was the potential data loss when sending and syncing a large amount of sensor data between Android handheld and Android Sony smartwatch 3. We will discuss the whole prototype to gain a higher sampling rate, solve the data loss problem and solve the synchronization issue in a later chapter of this thesis.

2. Accuracy: Accuracy of the sensor can be measured by the difference between the actual value and the measurement obtained by the sensor [4]. High accuracy means the measured reading is very close to the actual value. Post-processing, like data filtering, can be applied to suppress the noise in the raw sensor data, and make estimate with enhanced accuracy and reliability. We have designed data filtering techniques to achieve improved accuracy and precision of motion tracking.

2.3.3. Network Connectivity

Android Sony Smartwatch 3 provides power-efficient networking connectivity via 4.0 Bluetooth, Wi-Fi, GPS, NFC and USB [6]. Figure 2.4 illustrates how Sony smartwatch 3 connects as an IoT (Internet of Things) device through an Android handheld to the Cloud in order to transfer and sync a high volume of sensor data, to receive notifications, to send-receive messages, etc.

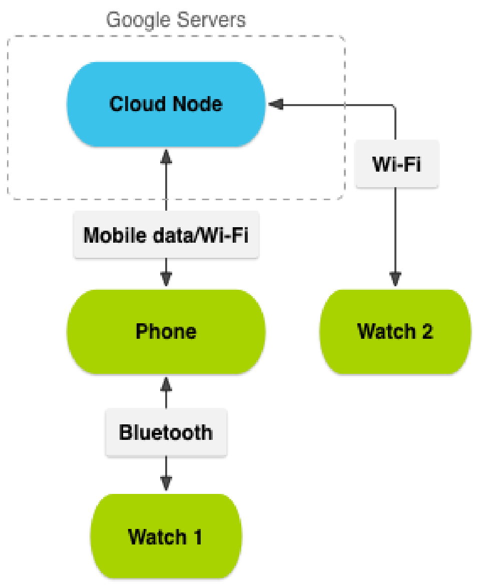


Figure 2. 4 Interaction between Android Wear and Android Handheld devices [8]

2.4. Android Sensor Problems & Limitations

In this section, we want to discuss the limitations and problems occurred when using raw sensor data extracted from Android Wear sensors. It is important to recognize and understand these issues when we develop Android based sensor system applications.

1. Human Error and Systematic Error: Human errors are mistakes made by humans while reading or extracting sensor data from a device. Systematic

errors, e.g., a constant offset inside raw sensor data reading, may affect the accuracy of a measurement, if not properly handled [4].

2. Random Noise: Random noise or errors, on the other hand, occur because there is a random noise inside raw sensor data measurement, and have to be handled differently, e.g., through statistical methods. Noise is an undesired, but unavoidable, signal present in the raw sensor signal. Without proper noise filtering, sensor raw data may appear meaningless. Various filtering techniques have been developed over the last century to deal with noise of different natures, such as white noise, Brownian noise, etc. [4].
3. Drift: Drift means some undesired data inside the raw sensor data that makes sensor raw data away from the correct values. Drift can happen due to sensor reading degrading over time. If we integrate sensor value, then there will be drift inside the integration result. There will be a constant value added to each iteration of the integration step which makes resulting sensor data reading drift away from the actual values [4].
4. Offset: When a device is in stationary situation, the value of motion sensor reading should be zero. Otherwise, there may exist an offset. For example, if device is not moving, then accelerometer sensor reading should actually looks like (0,0, -9.8 m/s^2). But, the measured sensor data is not exactly zero for x, y and -9.81 m/s^2 for Z axis due to bias or offset [4].
5. Sensor Time Delay and Data Loss: As we discussed earlier, higher sampling rate can capture higher-speed motion. But, at a higher sampling rate there may be increasing losses of data between two handheld and wearable. This

happens because Android is not a real-time operating system, and sometimes sensor delay may cause incorrect timestamp.

6. Integration Errors: To estimate correct position of an Android Wear, we need to double integrate Linear acceleration sensor data, which may cause integration errors. Mainly, we observed the following two type of errors [5].
    1. Acceleration Drift: When we estimate the position based on the acceleration data, each iteration of an integration may add constant offset into the resulting sensor data. This drift can be accumulated in the integration, which will result in a poor position estimate.
    2. Initial Conditions: To find positions based on acceleration, we may assume there is zero velocity and zero position initially.

In chapter 3, we will present a motion capture and analysis prototype using an Android handheld device. Then in chapter 4, we will discuss a novel motion capture and analysis system using Android Wear smart watch and their sensor capabilities.

Chapter 3

3. Prototype of a Stand-alone Motion Capture Device Using an Android Device

In this chapter, we will describe the use of Android handheld devices as a motion capture device. We will first review the sensor capabilities of Google Asus Nexus 7 Tablet device, and then our prototype and experimental studies.

3.1. Prototype Workflows

Figure 3.1 describes the workflow of our prototype of a stand-alone motion capture device that collects motion sensor data using an Android tablet.

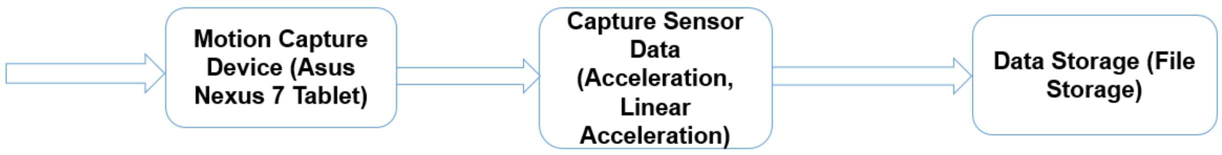

Motion Capture Device (Asus Nexus 7 Tablet)
Capture Sensor Data (Acceleration, Linear Acceleration)
Data Storage (File Storage)

Figure 3. 1 Workflow to collect sensor data from Asus Nexus 7 Tablet.

In this work, we have designed a prototype which can specify sampling rates and particular sensor type to capture data of interest in device coordinate systems and store corresponding timestamps, three-dimensional sensor data into a Microsoft Excel (.csv) file. This file is then used for further data analytics. Our motivation behind this is to design a Kalman filter base sensor data filtering algorithm to filter tablet linear acceleration sensor data and evaluate velocity and position. Later, use that algorithm to filter watch sensor data

3.2. Android Sensor API

Android Sensor API classes can identify and extract sensor data from device hardware. In this prototype, we have used the following classes [4]:

1. SensorManager: This class is used to create an instance of a sensor service, listing and identifying sensor list [7]. It is also used to register and unregister sensors. We can also set the sensor sampling rate and accuracy. Once sensor is registered, data will be extracted from the device hardware sensor chip.
2. Sensor: This class is used to create an instance of a particular sensor and their capabilities [7]. Sensor used to get important information about maximum range, minimum and maximum delay between two sensor event, sensor Name, Power, Resolution, manufacturer (vendor) Name, type and version.
3. SensorEventListener: It is used to receive notification when the sensor event is occurred [7]. This class provide information about sensor type, accuracy, timestamp, x, y, z three dimensional values array. When the event occurs, we are extracting accelerometer or linear acceleration sensor data timestamp and x, y, z values stored as one string for one event or sample [7].

1. Accuracy: describe sensor accuracy which refers to what percentage of sensor reliability or trust ability which is not in terms of reading are closely to the actual physical value or not.
2. Timestamp: Timestamp is a time generate at the time of the event occurs or sensor data value has changed which is in nanoseconds. But, later we are converting into milliseconds to find object position.
3. Sensor Values: As we discussed, device coordinates values for acceleration and linear acceleration sensor data. Values contains x, y, z three dimensional values.

3.2.1. Adjusting Sampling Rate

Sampling rate or sensor rates means number of sensor data samples extracting per second from Google Nexus 7. Sensormanager is used to register a sensor listener. So, after we register a sensor event listener for a given sampling frequency for a particular sensor, as soon as the sensor data becomes available from the hardware sensor, the event will be generated. Followings are the predefined rates in Android sensor API: SENSOR_DELAY_NORMAL, SENSOR_DELAY_UI, SENSOR_DELAY_FASTEST, SENSOR_DELAY_GAME [7]. However, it should be stressed that event may be delivered faster or slower than the specified rate due to the lack of strong real-time support in Android OS. Applications may choose different sampling rates, depending on its need. In our initial testing experiments, we have tested two parameters SENSOR_DELAY_NORMAL and SENSOR_DELAY_FASTEST for Nexus 7.

Alternatively, we can directly set the desired sensor delay between two sensor events (i.e., inter-sample time-interval) in microseconds for Android API level 9 and onwards which works for android Nexus 7 [7].

3.3. Capture Sensor Data

We have designed an Android application that selects sensor type between Acceleration and linear acceleration sensor and then chooses sampling rates SENSOR_DELAY_NORMAL, SENSOR_DELAY_FASTEST, or, alternatively, we can specify a sensor delay value in a microsecond. Fig. 3.2. shows the app GUI.

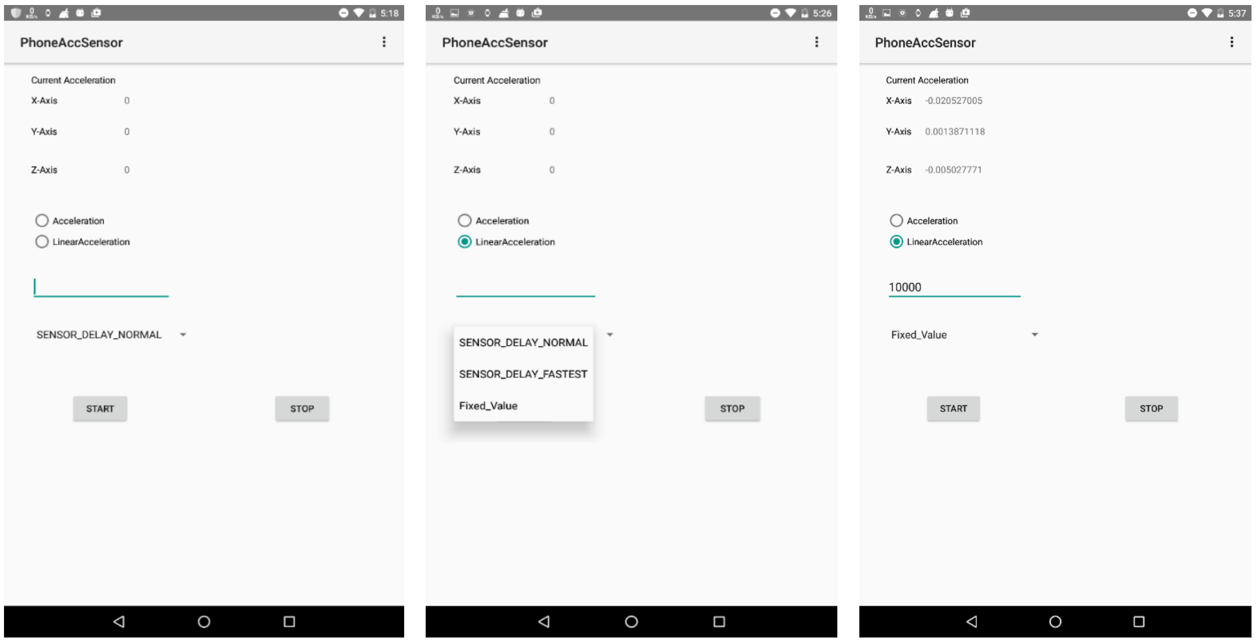


Figure 3. 2 Nexus 7 Android Prototype

3.4. Data Storage

According to our prototype sensor data which contains timestamp, three dimensional x, y, z values automatically store into a Microsoft Excel (.csv) file inside a tablet SD card. File storage is used to plot sensor data in graphical format as well as for further processing in sensor filtering and velocity, position estimation.

3.5. Experimental Studies

With this prototype, our goal is to demonstrate the various sampling rate capabilities and data sensory data quality and appropriate post-processing. We have carried out experiments for two different cases: i) when device is stably rested on a table, and ii) when device is in motion. We used SENSOR_DELAY_NORMAL sensor rates which is extract sensor data every 200 milliseconds, i.e., 5 samples per second.

Figure 3.3 presents the timestamp and sensor data for the Y-axis for case (i). Timestamp is given in milliseconds and linear acceleration sensor data on the Y-axis is in m/s^2. The inter-sample time-interval is calculated by the timestamp difference between two successive sensor events.

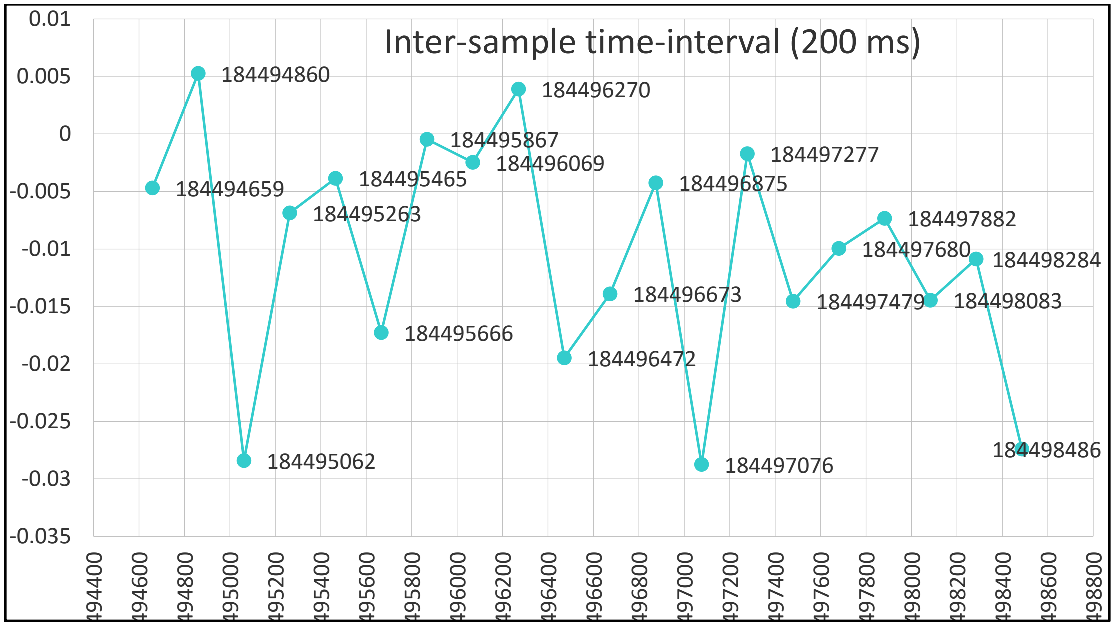


Figure 3. 3 Timestamp vs linear acceleration sensor data (200 milliseconds sensor rate)

For our purpose to estimate accurate velocity and position, it requires higher sampling rates at 100+ sample per second. Thus, we have started from SENSOR_DELAY_FASTEST which produce sensor samples around every 5 milliseconds. Figure 3.4 shows timestamp and sensor data on the Y axis with 5ms sensor rate.

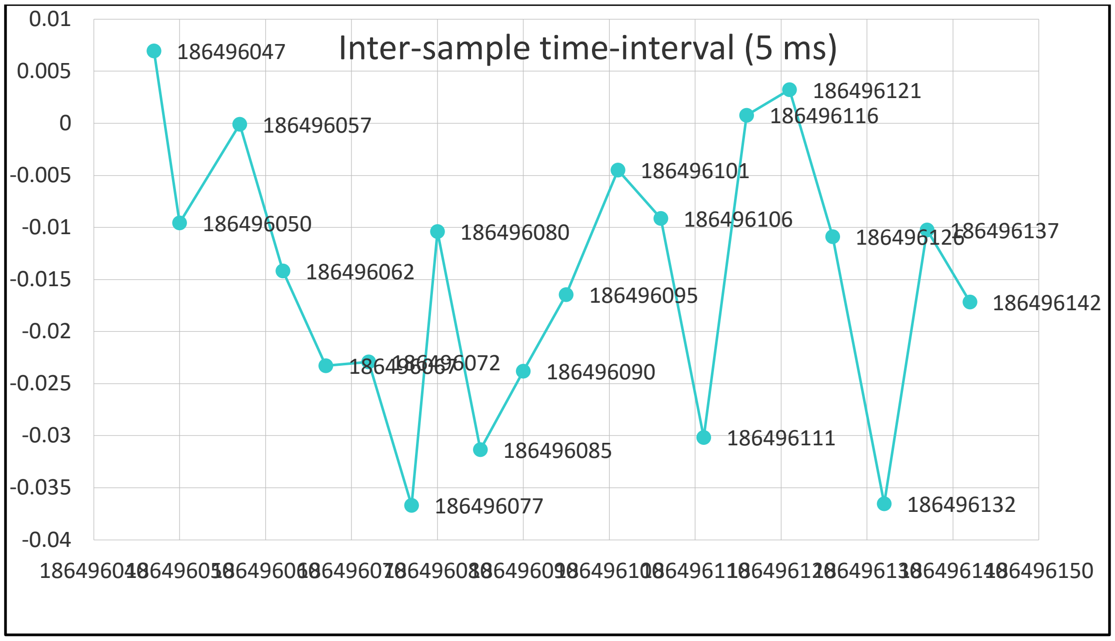


Figure 3. 4 Timestamp vs linear acceleration sensor data (5 milliseconds sensor rate)

However, 5 milliseconds appeared too high to produce stabilized samples sometimes. So, in our experiment, we set it to 10 milliseconds by explicitly entering the fixed sensor delay value in microseconds in our prototype. Figure 3.5 illustrates timestamp and sensor data on the Y axis with nearly 10ms sensor rate. Timestamp in milliseconds and linear acceleration sensor data on the Y axis in m/s^2.

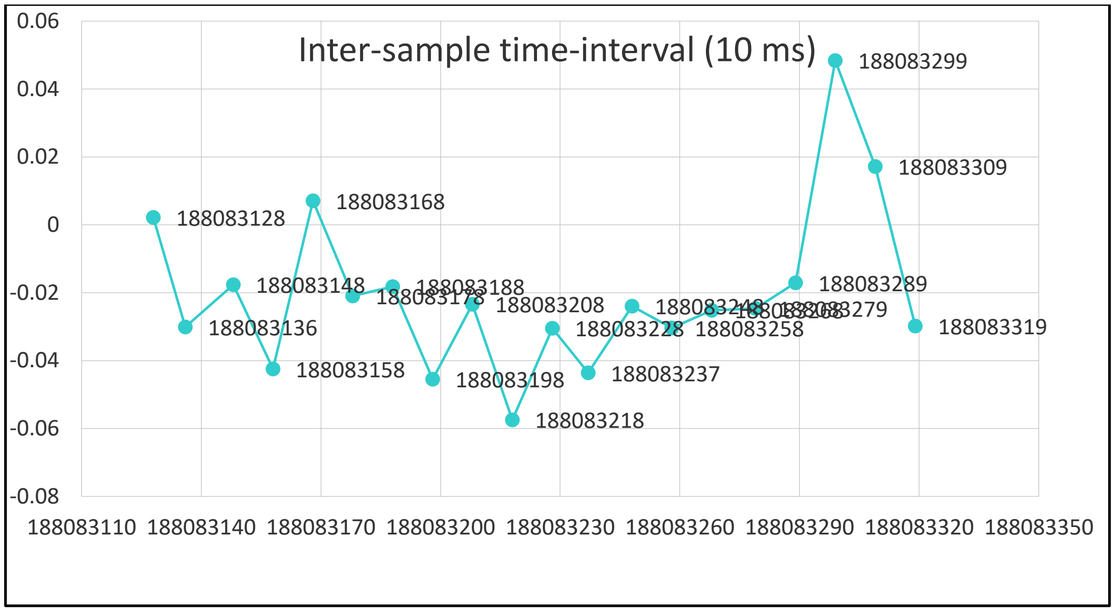


Figure 3. 5 Timestamp vs linear acceleration sensor data (10 milliseconds sensor rate)

3.5.1. Using Accelerometer Sensor

In Asus Nexus 7, the accelerometer sensor extract acceleration data with gravity portion which is exactly similar to the Sony smartwatch 3 but, the only difference is in Nexus 7 accelerometer, the sensor vendor is InvenSense. Figure 3.6 indicates acceleration sensor data captured from ASUS Nexus 7 when the device is stable on the table. So, the motion sensor data is closed to zero. Because of sensor errors, these data are not exactly zero. Z axis measures acceleration with gravity so it is close to 9.8 m/s^2 [7].

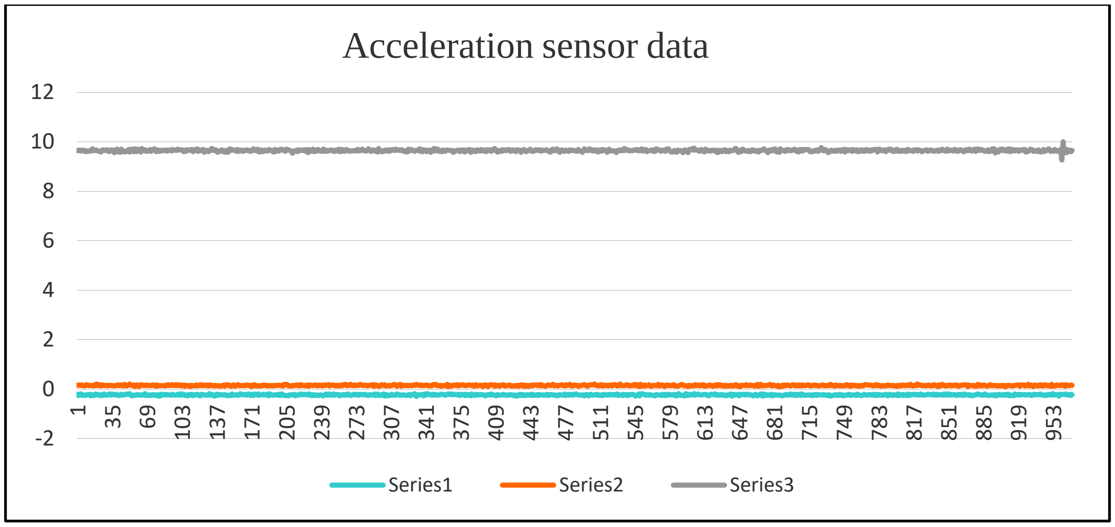


Figure 3. 6 Acceleration sensor data Series1-X Series2-Y Series3-Z

Figure 3.7 indicates acceleration sensor data captured from Asus Nexus 7 when the device is moving. Data is captured around x, y and z axes. Z axis values are close to 9.8 m/s^2 because of gravity included inside the acceleration.

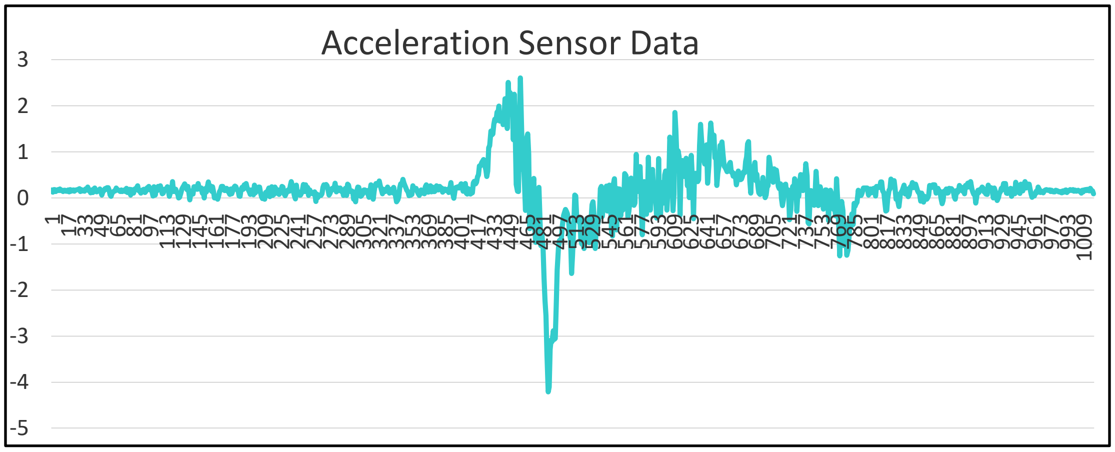


Figure 3. 7 Acceleration sensor data around device Y coordinate

3.5.2. Using Linear acceleration Sensor

Asus Nexus 7 linear acceleration sensor is similar to the Sony smartwatch 3 but the only difference is in Nexus 7 linear acceleration, the sensor hardware designed by Qualcomm. So, this sensor does not work as a secondary sensor which is exactly working as a raw sensor. Figure 3.8 indicates linear acceleration sensor data captured from Asus Nexus 7 when the device is stable on the table. So, the motion sensor data is closed to zero. Because of sensor errors, these data are not exactly zero.

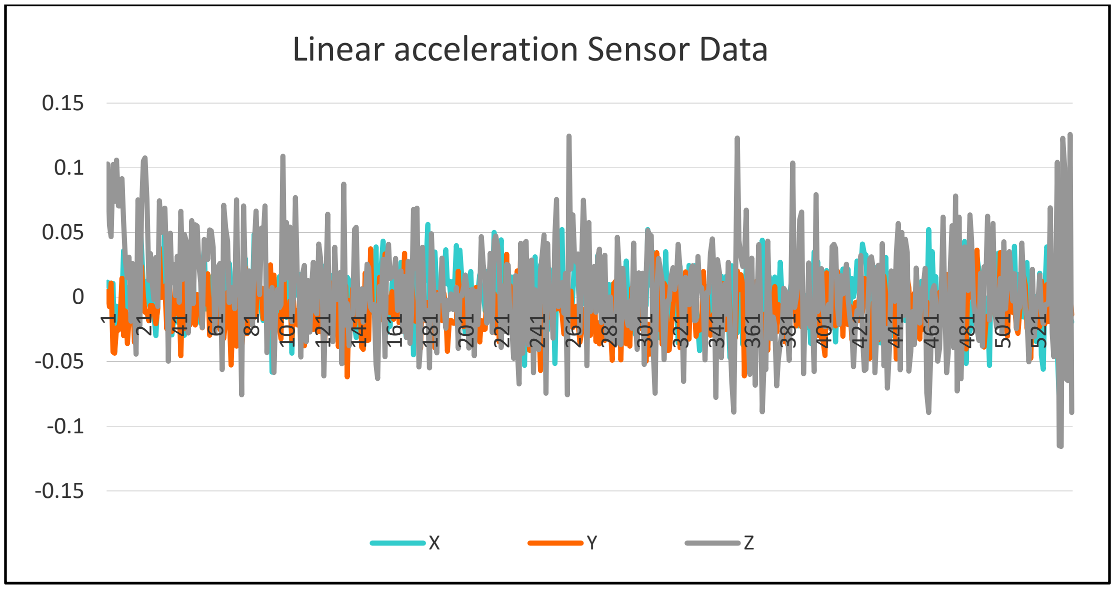


Figure 3. 8 Linear acceleration sensor data around device coordinates

Figure 3.9 shows linear acceleration sensor data captured from Asus Nexus 7 when the device is moving. One can clearly see the acceleration motion increasing and decreasing when device is moving otherwise motion values are close to zero.

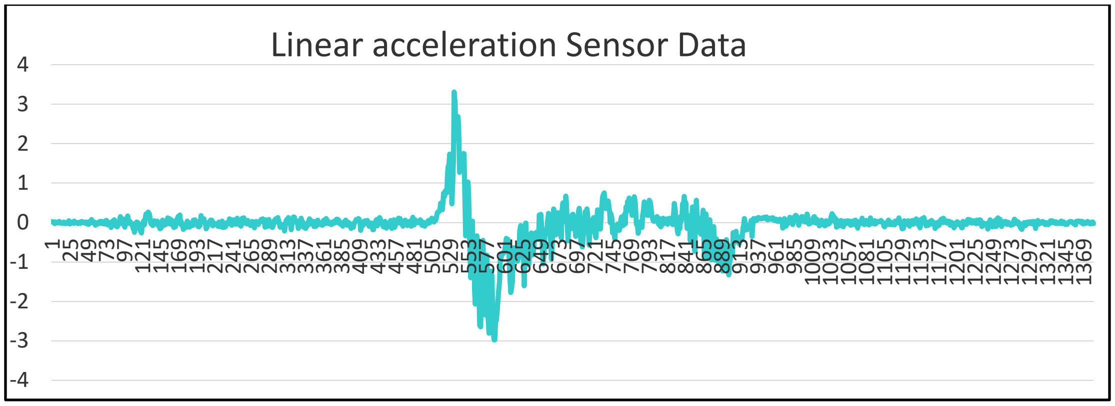


Figure 3. 9 Linear acceleration sensor data around device Y coordinate

In summary, in this chapter we introduced prototypes to collect raw sensor data using the Android Nexus 7 tablet. With the prototype system, we have collected the acceleration and linear acceleration sensor data and observed their noisy properties. This clearly shows the need for data processing to further enhance the accuracy of the motion data which we will discuss in Chapter 5.

Chapter 4

4. Sensor-Web based Cloud Solution and Prototype Designs

In this chapter, we will present our prototype design and describe the techniques to capture sensor data from an Android Wear motion capture device, and then transfer and store the collected data into the Cloud. The main goal of this research is to present a framework that enables large amount of networked sensors to collect and transfer data into the Cloud for real-time big data analytics and distributed collaborations.

4.1. Graphical Representation of Cloud based Prototype

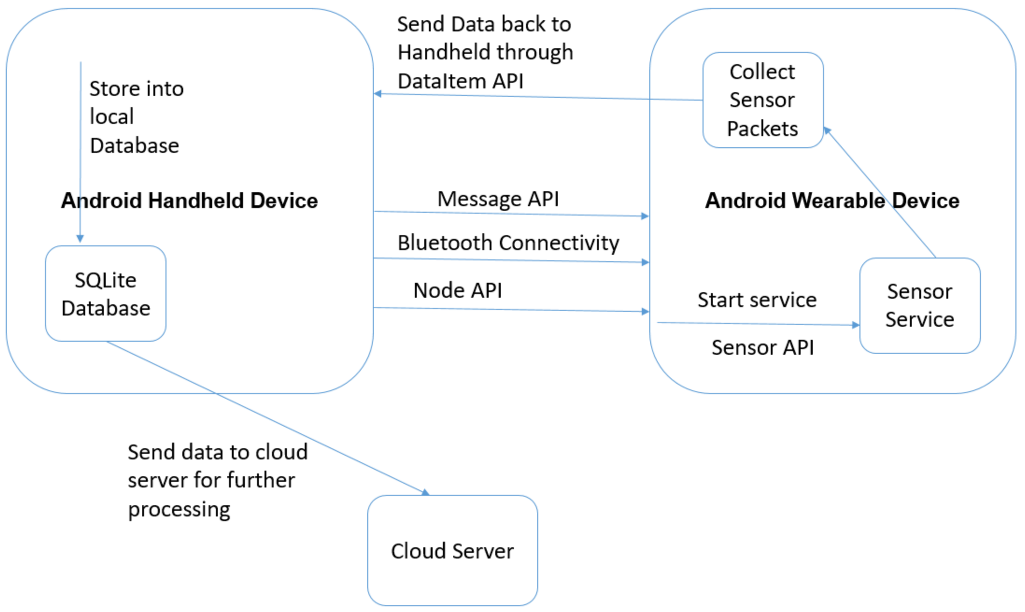


Figure 4. 1 Describe flow of our sensor-web based system prototype

Above, workflow describes the step-by-step work-flow of sending and synchronizing huge sensor data between Android handheld and Android Wearable device through Android Wearable APIs. It also provides mobile and cloud data connectivity to accommodate further sensor data storage and analytics.

4.2. Sony Smartwatch 3 Sensors

We have discussed in the previous chapter about all kind of built-in sensors available in the Sony smartwatch 3. In this work, in order to estimate object positions/motions, we will collect data from the following sensors from Android Wear. We have used both acceleration sensor data and linear acceleration sensor data [4].

1. Raw Sensors: Sensor.TYPE_ACCELEROMETER

   As we discussed in an earlier chapter, accelerometer sensor in the Sony smartwatch 3 measures acceleration applied to the device including the force of gravity. Sony smartwatch 3 measure acceleration along the device coordination system: X, Y and Z. During our initial testing experiments with Sony smartwatch 3 acceleration sensor, we observed that, when the watch rests stably on a table and with no acceleration, the X and Y axis acceleration readings are very close to zero while Z axis closely measure the force of gravity, which is 9.81 m/s^2. If one moves the watch toward its right, X acceleration value is positive; if one move the watch away from you, Y acceleration value is positive; and if one move the watch up towards the sky, Z acceleration value is positive.

Figure 4.2 presents the acceleration sensor data towards X, Y, Z axes when the watch rests stable on a table. We can clearly see that the Z-axis accelerations stay close to 9.8 m/s^2 because of gravity, while the other 2 axis accelerations are close to zero. It is also clear the sensor readings are noisy, which is as expected. Figure 4.3 presents the acceleration motion sensor data along the X, Y, Z axes when the watch is moving.

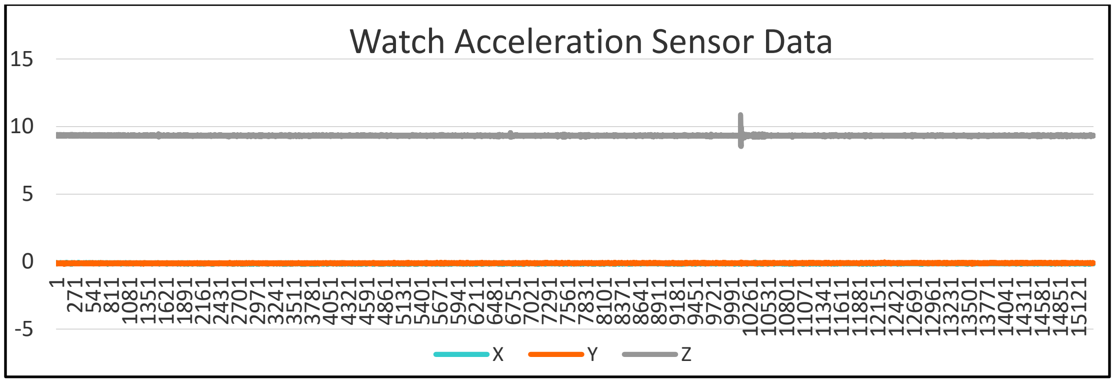


Figure 4. 2 Acceleration sensor data with all coordinates

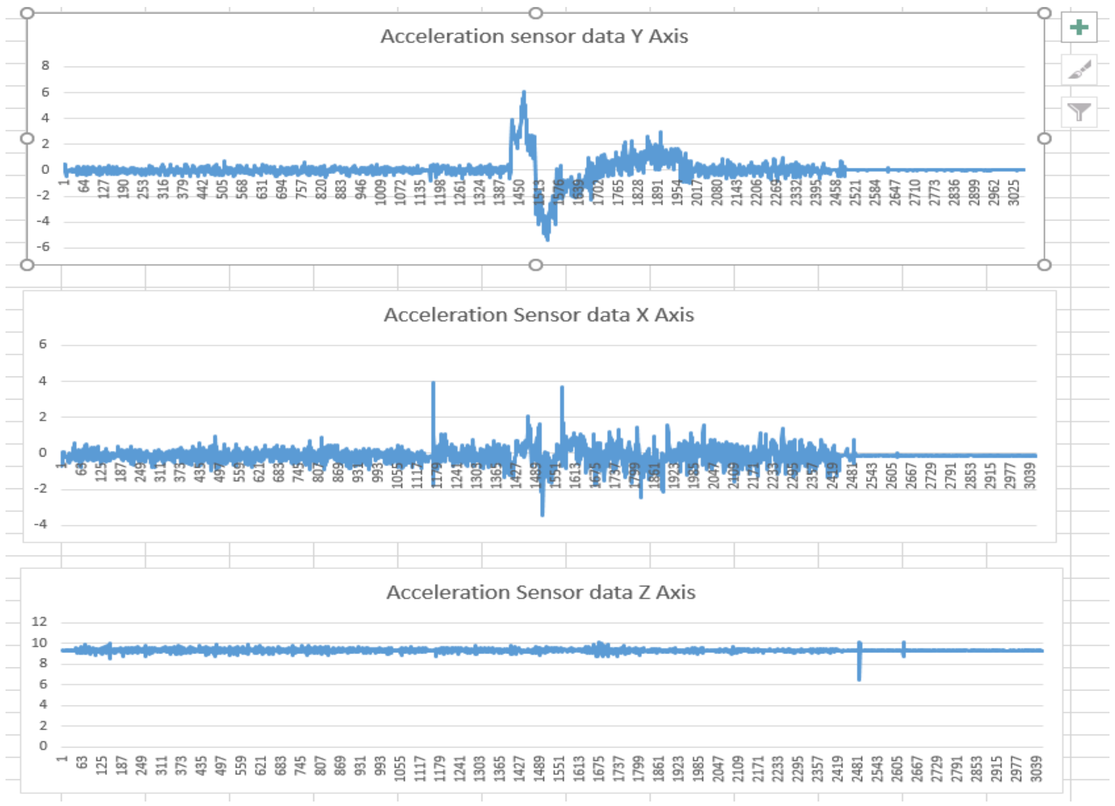


Figure 4. 3 Acceleration sensor data with all coordinates

2. Synthetic Sensor: Sensor.TYPE_LINEAR_ACCELERATION

In the Sony smartwatch 3, linear acceleration sensor is a synthetic sensor and is provided by AOSP (Android open source project). It measures the actual acceleration without gravity along the device 3-axis coordinate system. Linear acceleration sensor can also be used for further processing to estimate device position. Linear acceleration works the same as acceleration. Their only difference is the gravity. All measurements are in m/s^2 SI unit. One disadvantage of linear acceleration sensor is its offset, which we need to remove before using for further motion/position estimation.

In our prototype, we have designed lowpass-highpass filter solutions to filter out the gravity effect component from the acceleration raw data and to produce linear acceleration for our further processing.

We conclude that accelerometer sensor in Sony smartwatch 3 Android Wear is sufficient for measuring motion data to meet our need in this work. We will use the accelerometer sensor data to do further processing to estimate device position data.

4.3. Android Wear Sensor API

Android Wear Sensor API classes can identify and extract sensor data from device hardware. In this prototype, we have used the following classes [4].

1. SensorManager: This class is used to create an instance of a sensor service, listing and identifying sensor list [7]. It is also used to register and unregister sensors. We can also set the sensor sampling rate and accuracy. Once the sensor is registered data will be extracted from the device hardware sensor chip.

2. Sensor: This class is used to create an instance of a particular sensor and their capabilities. Sensor is used to get important information about maximum range, minimum and maximum delay between two sensor event, sensor Name, Power, Resolution, manufacturer (vendor) Name, type and version
3. SensorEventListener: It is used to receive notifications when the sensor event has occurred. This class provide information about sensor type, accuracy, timestamp, X, Y, Z three-dimensional values array. When the event occurs, we are extracting accelerometer or linear acceleration sensor data timestamp and X, Y, Z values stored as one string for one event or sample.
    1. Accuracy: describe sensor accuracy which refers to what percentage of sensor reliability or trust ability, which is not in terms of reading are closely to actual physical value or not.
    2. Timestamp: Timestamp is a time generated at the time of the event or sensor data value changed which is in nanoseconds. But, later we are converting into milliseconds to find object position.
    4. Sensor Values: As we discussed, the device coordinates values for acceleration and linear acceleration sensor data. Values contains X, Y, Z three dimensional values.

4.3.1. Adjusting Sampling Rate

Sampling rates or sensor rates means the number of sensor data samples extracting per second from Sony smartwatch 3. Sensormanager is used to register a sensor listener. So, after we register a sensor event listener for a given sampling frequency for a particular sensor, as soon as the sensor data become available from hardware sensor, the event will be generated. Followings are the predefined rates in Android sensor API: SENSOR_DELAY_NORMAL, SENSOR_DELAY_UI, SENSOR_DELAY_FASTEST, SENSOR_DELAY_GAME. However, it should be stressed that event maybe delivered faster or slower than the specified rate due to the lack of strong real-time support in Android OS. Applications may choose different sampling rate, depending on its need. In our initial testing experiments, we have tested two parameters SENSOR_DELAY_NORMAL and SENSOR_DELAY_FASTEST for Sony smartwatch 3. As shown, in Fig. 4.4 and 4.5, we have received sensor data every 60 milliseconds and 5 milliseconds, respectively. The inter-sample time-interval is calculated by the timestamp difference between two successive sensor events.

For our purpose to estimate accurate velocity and position, it requires higher sampling rates at 100+ sample per second. Thus, we have started from SENSOR_DELAY_FASTEST which produce sensor samples around every 5 milliseconds. However, 5 milliseconds appeared too high to produce stabilized samples sometimes. So, in our experiment, we set it to 7 milliseconds as shown in Fig. 4.6.

Alternatively, we can directly set the desired sensor delay between two sensor events (i.e., inter-sample time-interval) in microseconds for Android API level 9 and onwards which works for Android Wear Sony smartwatch 3.

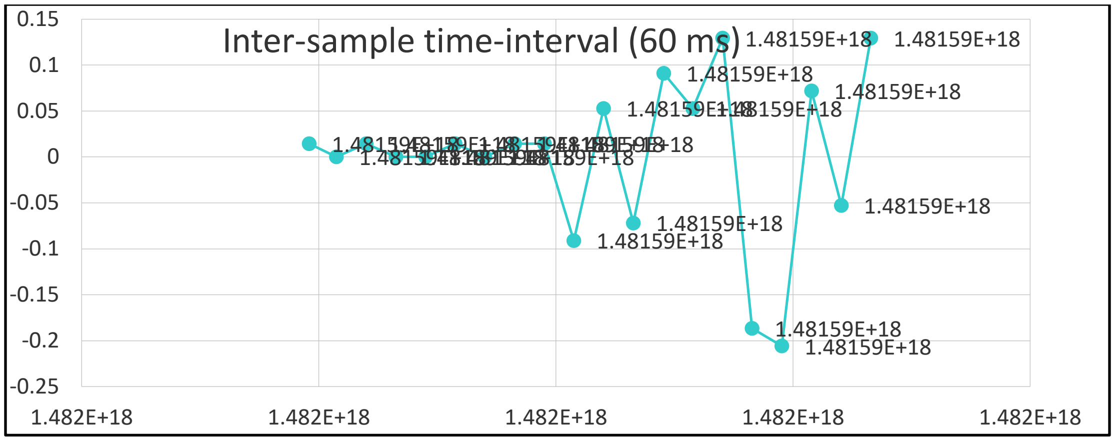


Figure 4. 4 Acceleration sensor data with 60 milliseconds sampling rate

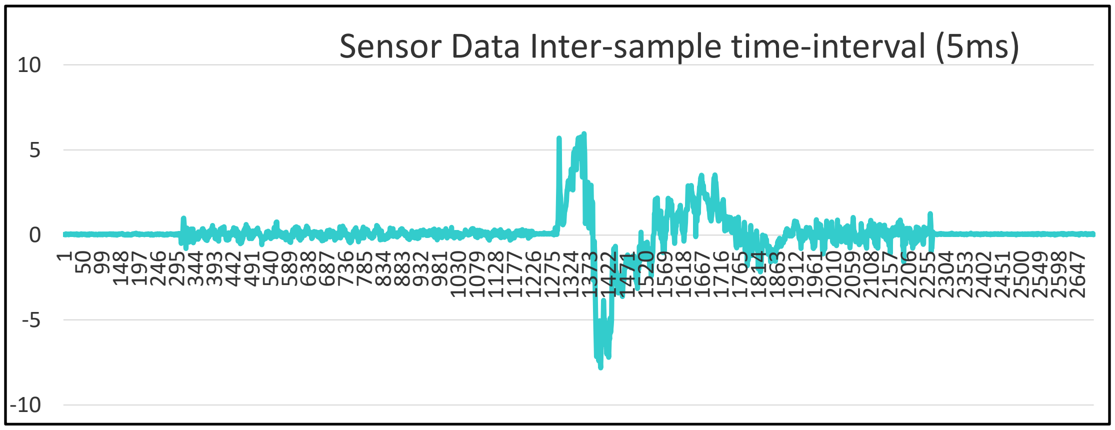


Figure 4. 5 Acceleration sensor data with 5 milliseconds sampling rate

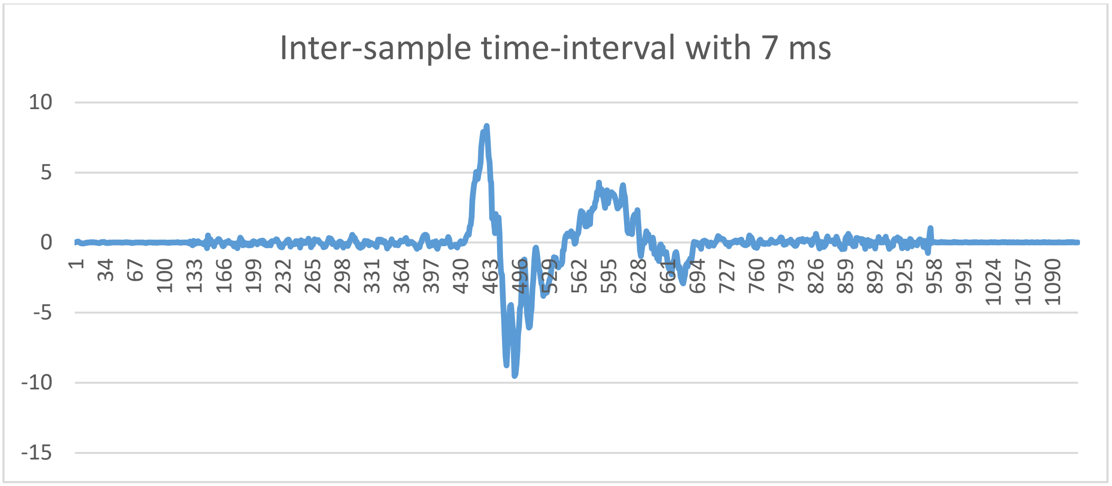


Figure 4. 6 Acceleration sensor data with 7 milliseconds sampling rate

4.4. Interaction between Android Wear-based Watch and Android handheld

In this work, we use Android Asus Nexus 7 as a handheld device and Android Sony smartwatch 3 as a wearable device for our prototype design. An operator will control the start and stop of a sensor activity by tapping on the Android handheld, which then will send an actuation message to the Android Wearable watch to notify the start/stop of a sensor service. Once actuated, the sensor service on the watch will collect accelerometer sensor data and send them from the Android Wearable to the android handheld device. In this section, we will discuss the interaction protocol design.

4.4.1. Established Connection between Android Handheld and Android Wearable

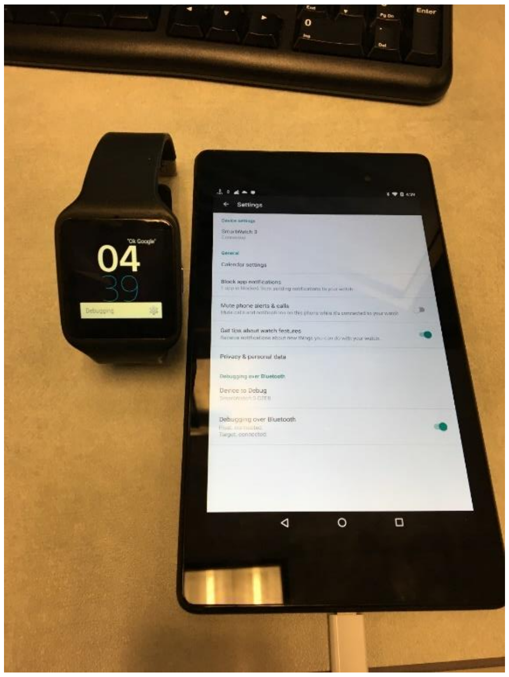


Figure 4. 7 The Bluetooth connection between the two nodes

As we discussed in the earlier chapter, Sony smartwatch 3 provides connectivity with other devices through Bluetooth 4.0. We use Android Studio IDE (Integrated development environment), Android Wear SDK (Software development kit) version 4.4 or later, and Android handheld SDK version 4.3 or later. Together, they provide us with the development environment and required libraries. Figure 4.7 shows that a successful connection has been established between Asus Nexus 7 and Sony smartwatch 3 by pairing, and both devices are ready to debugging over Bluetooth. After pairing Android Wearable with a handheld, the next step is to establish debugging over Bluetooth. This is completed by enabling USB debugging on the Android handheld device, and enabling ADB debugging and Debug over Bluetooth on the Android Wearable [9].

Following are the steps to correctly set up a Debugging Session [9].

1. Open the Android Wear Companion App.
2. Android handheld is connected with machine through USB.
3. Go to Android sdk platform tools and run following commands.

We can use any available port number to connect Android Wear device.

```
adb forward tcp:4444 localabstract:/adb-hub
adb connect 127.0.0.1:4444
```

4. In the companion app, if both Host and Target are connected then we successfully established connection between Android Wear and handheld for development purpose.

4.4.2. Sending and Syncing Sensor Data between Android Wear and Android Handheld

We will use Android Wear APIs to synchronize data between the Sony smartwatch 3, as a motion-capturing device, and the Nexus 7 Android Tablet, as a base station. In this section, we discuss our interaction and messaging protocols.

As illustrated in Fig. 4.1, for the actuation interactions, the operator will control the start and stop of a sensor activity by tapping on the Android handheld, which then will send an actuation message to Android Wear to notify the start/stop of a sensor service. In our prototype, as shown in the Fig. 4.8 The START TO SYNC DATA button is tapped to send a message from android handheld application to Sony smartwatch 3 to start collecting data, while STOP TO SYNC DATA button is tapped to send message from android handheld application to Sony smartwatch 3 to stop collecting data. We have used Android Wear Node and Message APIs to implement this messaging function.

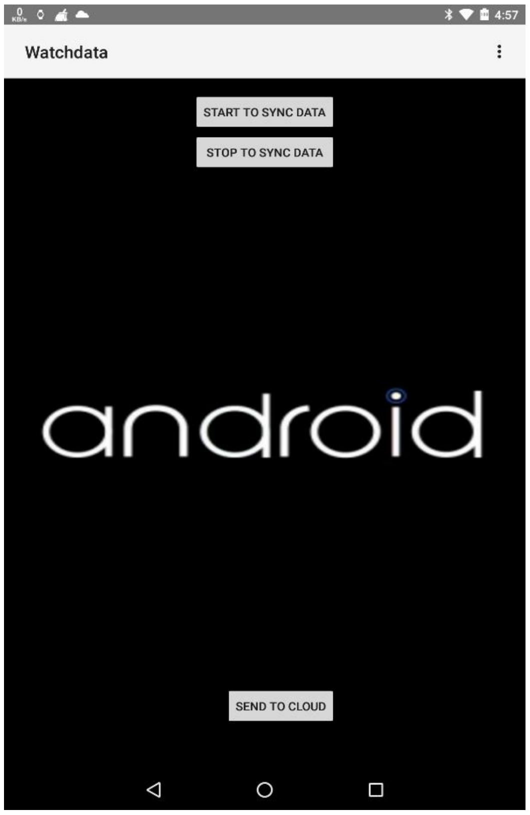


Figure 4. 8 Android handheld application design

4.4.2.1. Android Wear Node API

Before sending any message in a mobile/distributed environment, we first have to identify the local and the connected nodes. In Android Wear, Node API is used to identify local as well as all the connected nodes, such that we can deliver one message to all or one particular connected node [8].

4.4.2.2. Android Wear Message API

Android Wear worked in a way similar to RPC (Remote Procedure Call) which is used to request a service from another node or network [8]. Two items are attached with the message [11].

1. A path that uniquely identifies the message's action.
2. Message details.

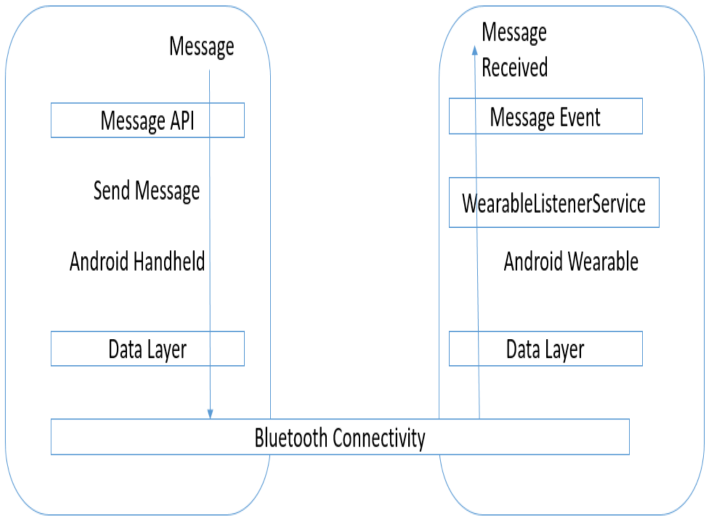


Figure 4. 9 Message transmission between two nodes

A unique path is important because multiple devices may connect with a user's handheld device simultaneously. Each connected device is considered as one slave node and has to be uniquely identified. In Android Wear Message, nodes are distinguished from each other through its unique path. As illustrated in Fig. 4.9, once a paired device is identified through Node API, the Android handheld can send a message to that node. On the other hand, on the watch, Message API is again used to receive the messages from the Android handheld. We will use a concurrent thread running in parallel for synchronous communication. Thread is a lightweight process used to perform a task in parallel and provide concurrency within the process.

4.4.2.3. Solution of data loss happens when higher sampling rate

Initially we had built a prototype that uses a 60 milliseconds sampling delay to collect sensor data and simultaneously send the sensor data sample to the handheld side through parallel thread. The system worked without any issue. However, when we lifted the sampling rate higher, we observed frequent data loss. This was due to the resources consumed by the increasing number of threads running in parallel when the sensor events became much more frequent.

To address this issue, we redesign the data transfer protocols to limit the number of concurrent threads. Instead of processing and sending the sensor data upon the notification of each sensor event, we make use of local storage at the watch by cumulating multiple samples of sensor readings and then making one packet to send to the handheld from the watch. We make use of the Android Data Item API to manage the storage and synchronize sensor data between the nodes. Data Item consists of following entities [10]:

1. DataMap: Datamap is a key-value pair data structure contains a key which represents datamap and value contains 20 samples buffer. It works as a byte array and the limitation of dataitem API is 100 KB of data storage while sending and syncing;

2. Path: A unique path to identify the node on other side.

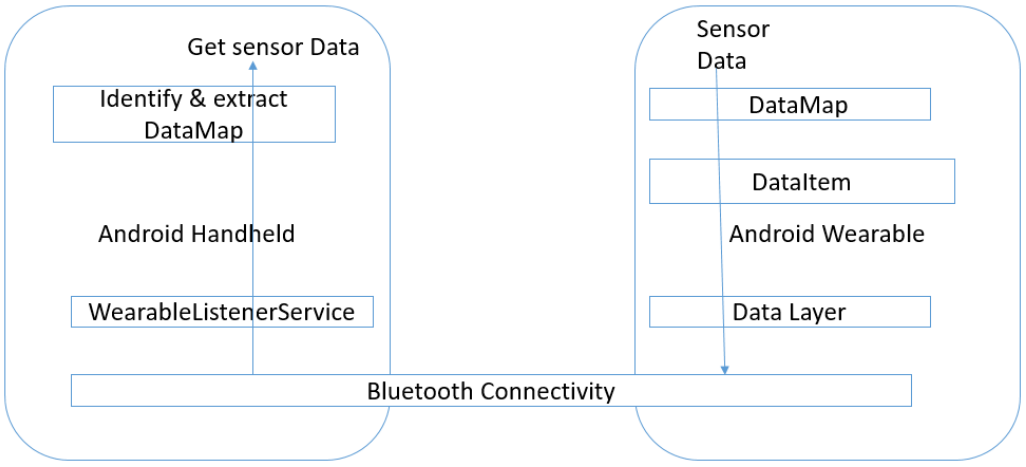


Figure 4. 10 Syncing data from Android Wear to handheld through DataItem.

As illustrated in Fig. 4.10, extracted sensor data from Android Wear are first locally stored into DataMap, and then, in groups, sent to Android handheld in real-time. The number of samples being grouped together has to be carefully considered in order to limit the transfer delay for real-time applications. On the other hand, the receiver at the handheld will extract the data from DataMap and store it into the local database.

4.5. Interaction Between Android and the Cloud

Android handheld receives sensor data and then save the data to its local SQLite database for storage purpose. As illustrated in Fig. 4.1, in order to augment the sensory system processing and storage capabilities, we will attach the sensor system to the cloud to construct a sensor web system. To sync data to the cloud-based service, we will use the Android asynchronous task APIs which will send data to the cloud concurrently in the background, whereas the main process is simultaneously receiving sensor data from the Android Wear. Moreover, the sensor-web based cloud-solution has the following advantages:

1. Easily synchronize large amount of sensor data between local storage and cloud storage.
2. Accessible from anywhere and anytime for any authorized applications.
3. Able to perform efficient insert, update, delete and extract operations for sensor data use.
4. Can support potentially large number of sensor systems simultaneously by using transactions to deal with concurrency control in the common dataset accesses.
5. Provide unlimited computing power for further sensor data Analytics and processing.
6. Provide Data Backup and recovery facility in face of failures.
7. More secure, scalability, reliable and cost-efficient.

4.6. Prototype Advantages

Throughout this chapter, we discussed our prototype to collect sensor data at real-time from Sony smartwatch 3 and store it into cloud database. Following are the advantages and results of this prototype.

1. Successfully sending and syncing large amount of sensor data between Android Wear and android handheld device.
2. Extracting a raw Accelerometer sensor data in real-time from Android Wear.
3. Avoid data loss issue.
4. Provide local as well as cloud storage solution to store big sensor data for further data analytics purpose.
5. User can wear android Sony smartwatch 3 and connect with android handheld, through this prototype capture motion sensor data and store into cloud to further process to measure acceleration, velocity and position data of user movement.

Chapter 5

5. Estimate Acceleration, velocity and Position through sensor data Filtering techniques

In this chapter, we will discuss sensor data noise and errors, and present our cloud-based sensory data processing techniques in detail. Specifically, we will start with discussing the sensor noise and errors presented in raw sensor data, and then provide a Kalman filter based signal estimation solution to obtain enhanced-precision acceleration, velocity and position data by mitigating the noisy effect of acceleration sensor, and the cumulative errors during the integration process to obtain velocity and position.

5.1. Observed Problems in Collected Sensory Data

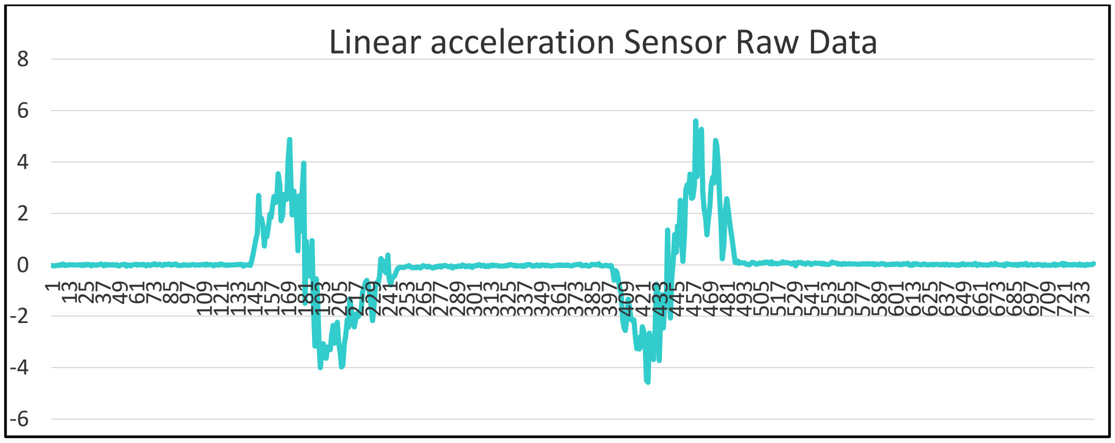


Figure 5. 1 Linear acceleration Sensor Y Axis raw noisy data with errors.

As illustrated in Figure 5.1, Linear acceleration raw sensor data collected from Google Nexus 7 may be effected by the following errors [4].

5.1.1. Raw Sensor Data Errors

Major errors observed in the raw sensor data include:

1. Noise: As shown in Fig. 5.1, linear acceleration sensor data are clearly noisy. Noise is a random fluctuation of measured sensor data and carries no useful information, which has to be removed by processing in order to obtain the actual data of interest. For motion sensor, noisy sensor data may be caused by vibration, temperatures, etc. So, without proper filtering, this kind of undesirable signal sensor data produce inaccurate results of acceleration, velocity and position.
2. Drift: Drift moves sensor data away from the real value. Linear acceleration raw sensor data has drift in its readings. Drift may be occurring due to sensor degradation occurring over time. Also drift is presented after integrating acceleration data to obtain velocity and then positions.
3. Offset or Bias: Linear acceleration raw sensor data always has an offset inside that, as illustrated in Fig. 5.2. Clearly, sensor data x, y, z axes values are not exactly zero when at rest on a table, indicating constant offset in the readings.

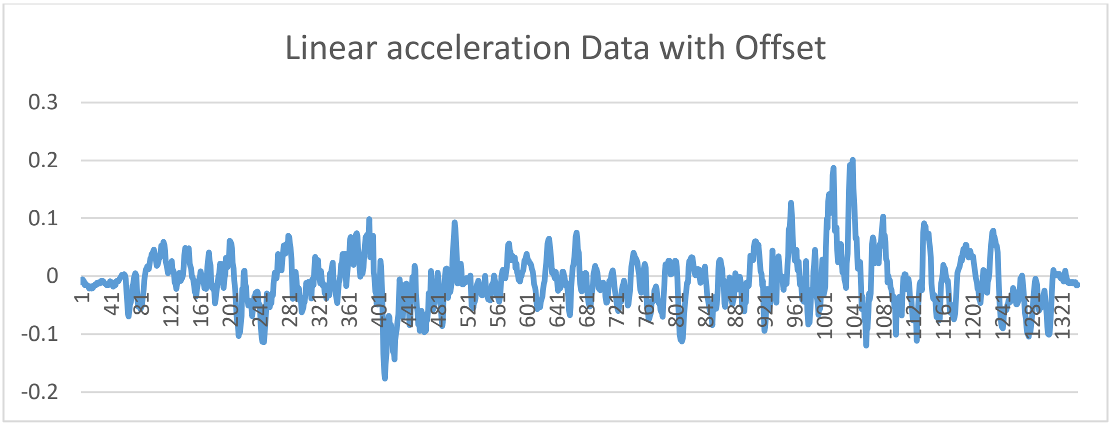


Figure 5. 2 Linear acceleration Data with Offset/Bias Error

5.1.2. Integration Errors

Initially, we tried to integrate raw linear acceleration sensor data to get velocity and again through integration to find position. Unfortunately, the initial experimental results are not accurate because of following reasons [5]:

1. Linear acceleration sensor data has drift inside, resulting in distorted estimates when we integrate them to generate velocity. In such cases, even a small offset will cumulate at each iteration of integration to make data drift away from real value quickly.
2. The Sampling rate of motion sensors is not consistent through the experiment, resulting in a mismatch when the processing algorithm assume the same inter-sample delays. Furthermore, for fast motion, a low sampling rate limits its capabilities to track the fast movement. This requires the use of a higher sampling rate, which, unfortunately, causes even more fluctuations in the sampling rate.

3. To use linear acceleration sensor data to calculate velocity and position, we must know the initial velocity and position which cannot be made available through the motion sensors alone.

5.2. Overview of Solution

In this section, we have designed two filtering techniques low-pass/high-pass filtering and Kalman filter techniques. Low-pass/high pass filtering is used to remove gravity component from collected raw acceleration sensor data [12]. Kalman filter is used to mitigate the random noise and estimate the corresponding acceleration, velocity and position from the filtered acceleration sensor data [13].

5.3. Low Pass/High Pass Filter

In our Sony smartwatch 3 based sensor data collection prototype, acceleration sensor readings include the gravity. So, we have used low-pass/high-pass filter to filter out the gravity effects from raw acceleration sensor data [12]. Specifically, low pass filter is used to identify the force of gravity and then high pass filter is used to subtract that gravity data from raw acceleration sensor data. Here we are producing synthetic linear acceleration sensor using raw acceleration sensor data and the filtering technique.

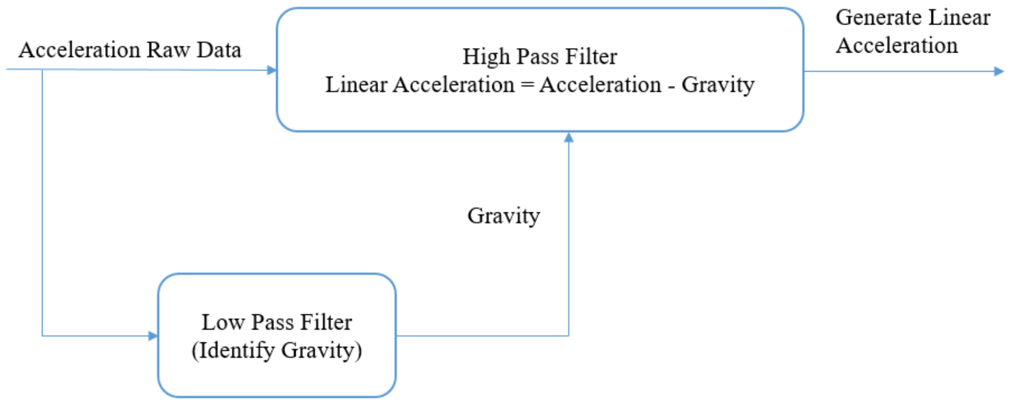


Figure 5. 3 Block Diagram of Low-pass and High-pass Filtering

Low-pass filter passes signal with the frequency lower than the cut-off frequency. To choose cut-off frequency we are using α parameter. The α value is calculated by the following formula [12].

$$\alpha = t / (t + dt),$$

Where *t* is the low-pass filter time constant and *dt* is the sampling rate. When filtering Sony smartwatch 3 sensor data, *dt* (sampling rate) is around 5 milliseconds, and *t* depends on the latency the filter adds into the sensor event, which is 20 milliseconds. As a result, $\alpha = 0.83f$ in our case to identify gravity from raw acceleration sensor data.

Once gravity is identified by the low-pass filter, the high-pass filter is used to remove it from raw acceleration data. High pass filter passes signals with frequency above the cutoff frequency. Here we are using this filter to subtract from raw data. Fig.

5.4 presents the raw acceleration sensor data and linear acceleration sensor data produced by the low-pass/high-pass filter.

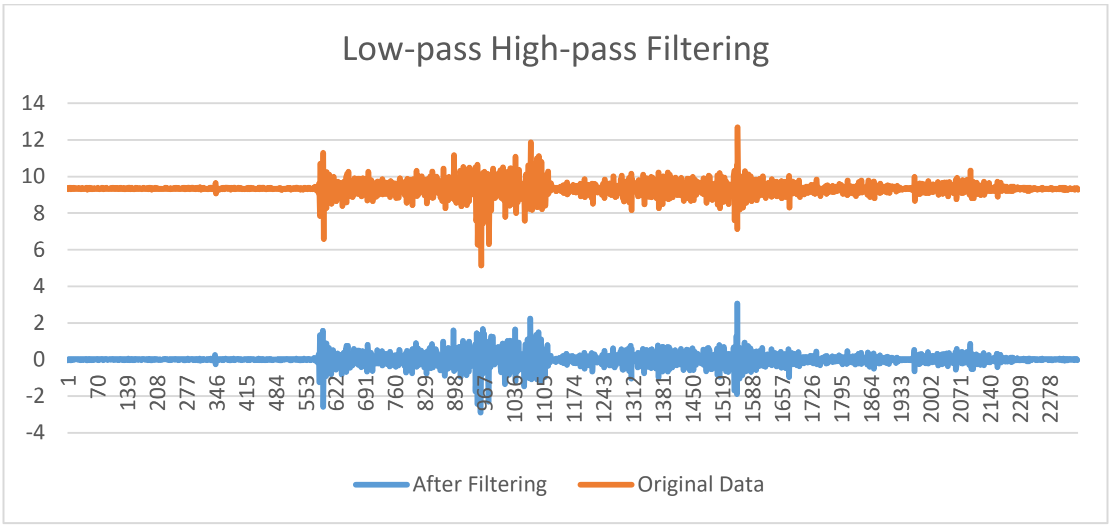


Figure 5. 4 Linear acceleration Data before and After Low-pass High-pass Filtering

5.4. Kalman Filter based De-noising and Motion Data Estimation

We have designed Kalman filter-based sensor data analytics prototype running in the cloud to filter sensor noise and drift, remove and/or minimize sensor and integration errors in order to calculate acceleration, velocity and position. Figure 5.5 presents the workflow of our system operations in detail.

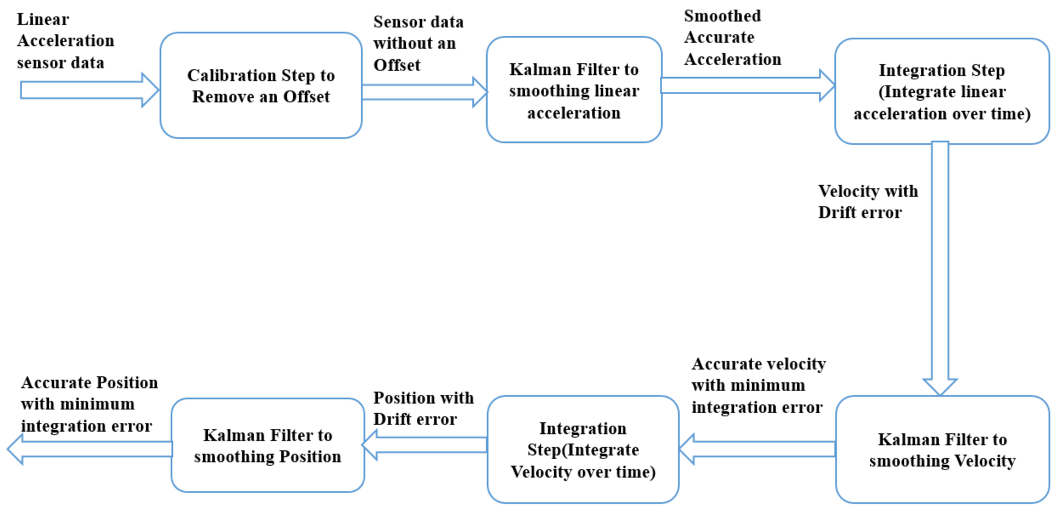

Linear Acceleration sensor data
Calibration Step to Remove an Offset
Sensor data without an Offset
Kalman Filter to smoothing linear acceleration
Smoothed Accurate Acceleration
Integration Step (Integrate linear acceleration over time)
Velocity with Drift error
Kalman Filter to smoothing Velocity
Accurate velocity with minimum integration error
Integration Step(Integrate Velocity over time)
Position with Drift error
Kalman Filter to smoothing Position
Accurate Position with minimum integration error

Figure 5. 5 Workflow of acceleration, velocity and position estimation using Kalman Filter

5.4.1. Calibration Step to Remove an Offset

As we discussed earlier, even when the device is stable on the table, the observed x, y, z coordinates sensor readings are not exactly or close to zero due to the offset within the linear acceleration sensor data. We have to remove it to obtain the correct measures. Moreover, without removing the offset, the cumulative nature of the integration steps to calculate the velocity and position will further worsen the problem and lead to in-correct results. Thus, we have designed techniques to remove an offset from raw sensor data [4]. In our prototype, we have applied the following steps to remove an offset.

1. Take a number of samples while the device is static.
2. Estimate the offset at each axes x, y and z based on the collected samples.

3. Then subtract the corresponding estimated offset from each measured value at the particular axis.

Figure 5.6 presents the sensor data before and after performing calibration.

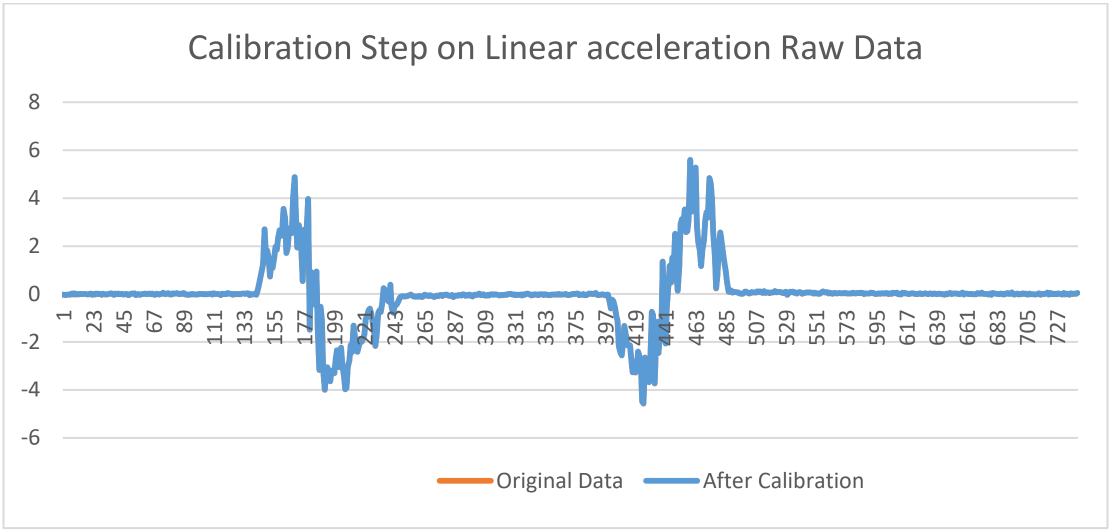


Figure 5. 6 Linear acceleration Sensor Y Axis sensor data before and after removing Offset

5.4.2. Kalman Filter

We have started with using simple low pass filter to denoise the collected data. However, the performance is very limited. So, we decided to investigate the performance of using Kalman filter for motion data processing. Kalman filter is a data processing algorithm that filter out noisy sensor data and produce improved estimate of data being observed, which has been widely used in digital signal processing [13]. It is generally more complicated processing algorithm compared to other low-pass filtering algorithm, resulting in higher computation demand, particularly for high-dimensional signal processing applications. However, this is not of major concern for our application as our motion data are of limited dimensions and we only have to deal with limited order of matrices. Furthermore, we have designed a multi-staged Kalman filtering approach to further reduce the computational complexity. Our prototype implementation and experimental studies have demonstrated it as highly suitable for real-time processing.

5.4.2.1. Implementing Kalman Filter

Kalman filter can be used to predict the state of a system when there is a lot of noise in the input sensor data. To implement the Kalman filter, we need to understand all the matrices, parameters and their effect on removing noise and correct error, thus producing more accurate results. In particular, we apply the Kalman filter based algorithm to filter out noisy linear acceleration sensor data to get an accurate acceleration. We also apply the Kalman filter to filter and minimize integration errors after each integration in our algorithm to calculate the corresponding velocity and position.

We have designed a three-dimensional Kalman filter to filter out noise over all the three coordinates. To implement the three dimensional Kalman filter, we follow these two steps: prediction and corrections [13]. Each step involves different matrix operations as described below [13].

1. Prediction Step: Projecting the state ahead of your current stage is called prediction. For instance, when one wants to filter the acceleration, velocity or position data, first this step will be called:

X0 = F. times(X). plus (B. times(U));

P0 = F. times(P). times (F. transpose ()). plus(Q);

2. Correction Step: We will pass the measurement matrix or input sensor data output from previous prediction step to our algorithmic workflow, producing an accurate estimate of the corresponding motion state as the output.

Matrix S = H. times(P0). times (H. transpose ()). plus(R);

Matrix K = P0. times (H. transpose ()). times (Inverse ());

Matrix Y = Z. minus (H. times(X0));

X = X0. plus (K. times(Y));

Matrix I = Matrix. Identity (3,3);

P = (I. Minus (K. times(H))). times(P0);

Below is a description about each matrix and parameter used in the above equations. Also, all the values of these matrices and parameters we have selected based on our experimental results.

F - state transition matrix
X - state vector
B - input gain or control input matrix
U - input vector
P - error covariance matrix
Q - process noise covariance matrix
X0 and P0 - output of prediction step
H - measurement matrix
R – measurement noise covariance matrix
Z - measurement vector as an input unfiltered sensor data
S, K, Y - intermediate matrices variables to store equations output value for further usage.

5.4.2.2. Selecting the Kalman Filter Parameters

Initially, we have used 6x6 (combination of either acceleration & velocity, or velocity & position) and 9x9 (combination of acceleration & velocity & position) Kalman filter. However, after checking all cases, we have reached to a 3-stage 3x3 (progressively, in the order of acceleration, velocity, position) Kalman filter design. Following are the values and parameters used for the matrices and vectors [13].

1. F - The state transition matrix is a 3 rows and 3 columns matrix which initially set to {{1,0,0}, {0,1,0}, {0,0,1}}.
2. X - The state vector represents the output of the Kalman filter after each correction step and is further used to predict the next state value and correct the next sensor data samples. The state vector is initialized to all-zero vector and then updated every time in the Kalman filter process. For instance, the initial value of this vector is {0,0,0} which is indicate of *{Ax, Ay, Az}* – the accelerations along X, Y, Z axes. Then its values are updated accordingly at each filtering round.
3. B – The input gain or control input matrix which is set to 3x3 identity matrix.
4. U - The input vector which is set to {0,0,0}.
5. P – The error covariance matrix initially set to 3x3 identity matrix. P is updated at each round for the estimated error.
6. H – The measurement matrix which is 3x3 identity matrix.

7. Z – The measurement vector as an input unfiltered sensor data. This vector works as an input sensor data. Pass the input sensor data sample x, y, z as a measurement vector in the Kalman filter correction step.
8. Q - Process noise covariance matrix which is a 3x3 matrix that characterizes the process noise.
9. R – The measurement noise covariance matrix which is a 3x3 matrix that characterizes the sensor noise.
10. K – The Kalman gain in above equation.

5.4.2.3. Selecting the Kalman Filter Parameters for Linear acceleration Data

Now, we will look at the Kalman-filtered linear acceleration data with different values of Q and R [14]. It is a challenge to choose the appropriate values for process noise Q and covariance noise R. However, sensor data smoothing and accuracy is sensitive to and depending on proper choices of Q and R matrices.

In our following experiment assessment, we have used different value settings for the Kalman filter, and presents the results in Fig. 5.7-10, and conclude that the values of, Q = 0.0625 and R = 4, have shown the best performance, which will set [14]:

Q matrix = {{0.0625,0,0}, {0,0.0625,0}, {0,0,0.0625};

R matrix = {{4,0,0}, {0,4,0}, {0,0,4}}.

First, Figure 5.7 presents the filtering results with Q = 128 and R = 10.

1. Q = 128 and R = 10

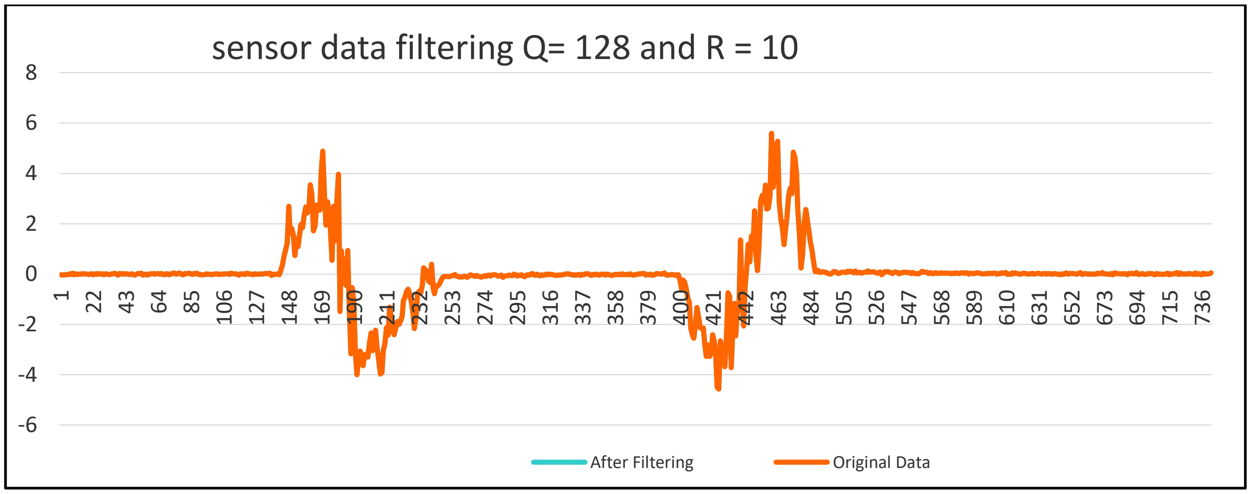


Figure 5. 7 Linear acceleration Sensor Y Axis sensor data filtering with Q=128 and R=10

There is no impact on the data by the filtering. Both original and after filtering sensor data are the same. Then, we reduce Q to 4.

2. Q = 4 and R = 10

As shown in Fig. 5.7, the Kalman filter with the new setting show improved de-noising performance, but still significant noise exists. Then, we continue the trend and set the values to (Q = 0.125 or 0.0625). As shown in Fig. 5.8 and 5.9, the decreasing value of Q lead to smoother results in the sensor data. However, we also can observe increasing lag between the actual sensor readings and the filtered results, which may cause unnecessarily high latency in the tracking system.

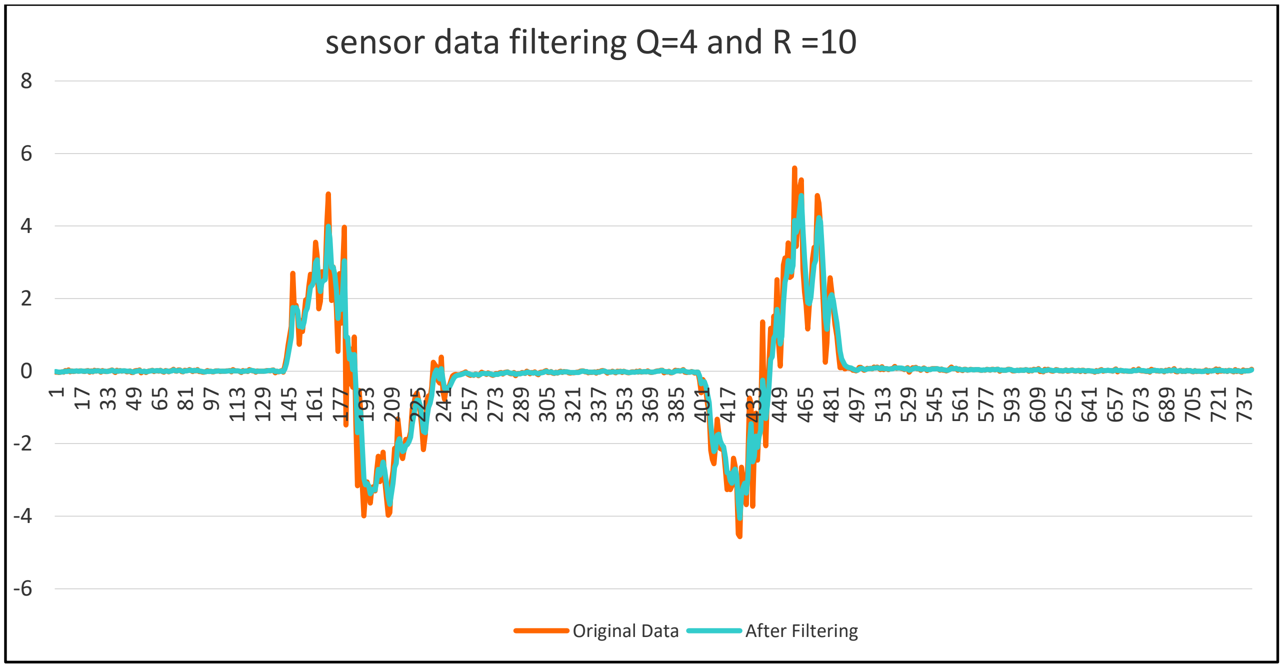


Figure 5. 8 Linear acceleration Sensor Y Axis sensor data filtering with Q=4 and R=10

3. Q = 0.125 and R = 10

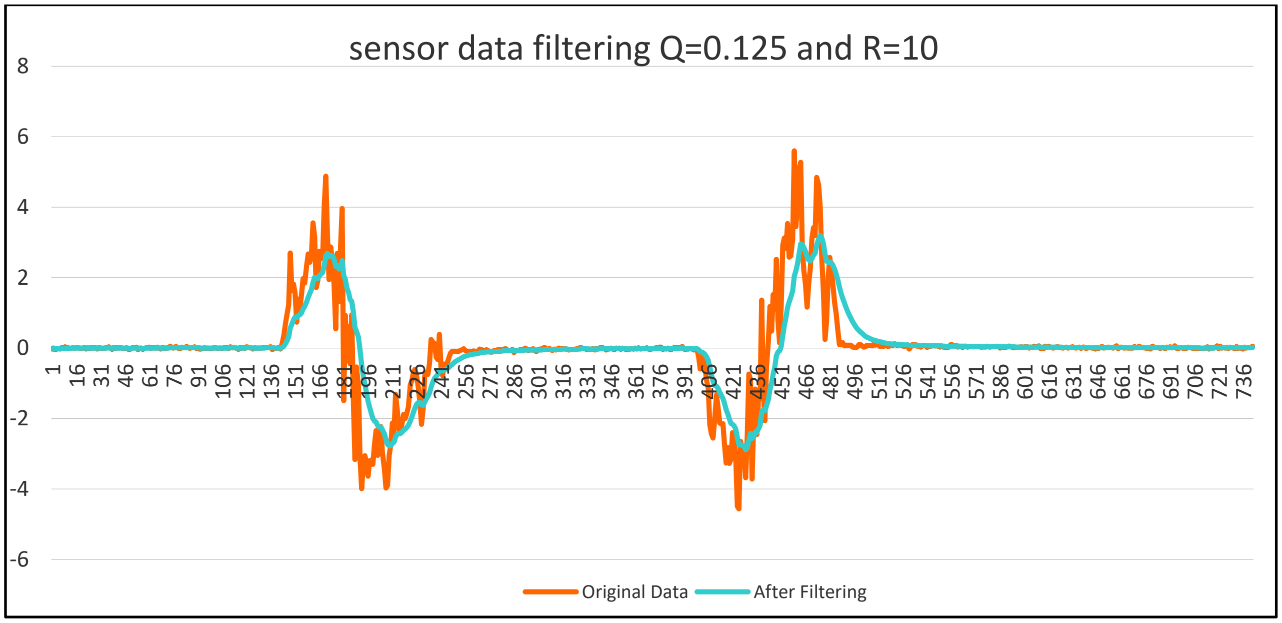


Figure 5. 9 Linear acceleration Sensor Y Axis sensor data filtering with Q=0.125 and R=10

4. Q = 0.0625 and R = 10

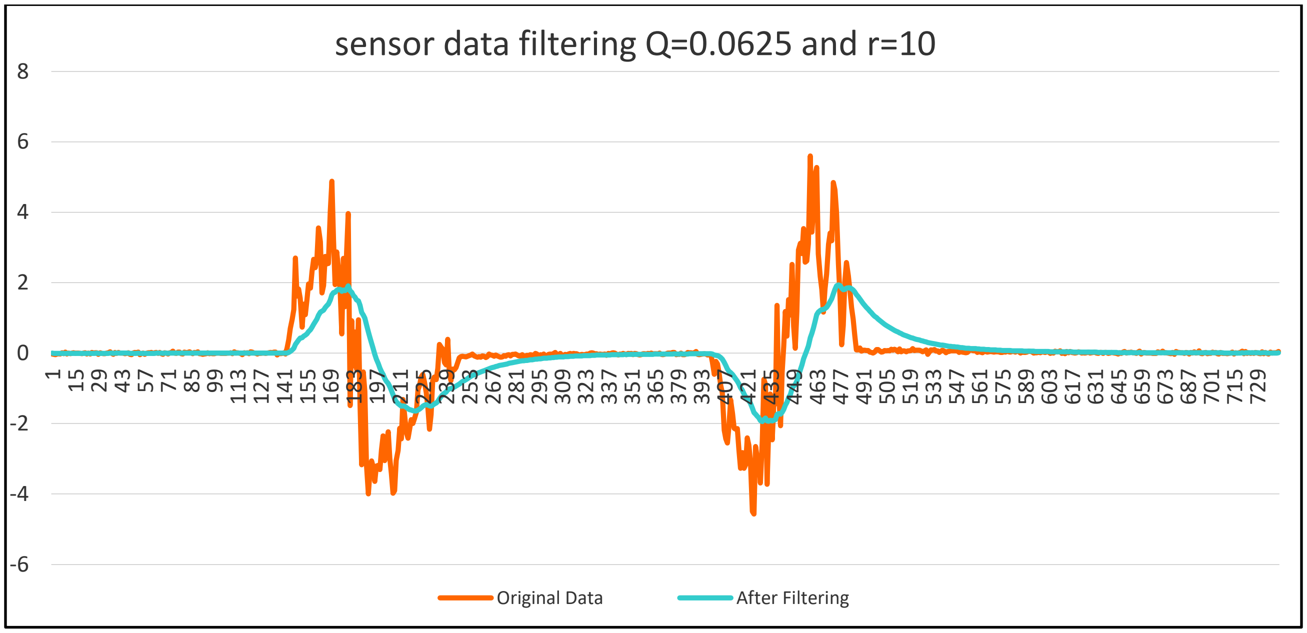


Figure 5. 10 Linear acceleration Sensor Y Axis sensor data filtering with Q=0.0625 and R=10

To limit the lag between the actual sensor readings and the filtered results, we reduce the value of R to give more weight to the actual sensor samples in the filtering process. As shown in Fig. 5.11, the resulting data show a better balance between the de-noising and the sensor samplings as the filtered curve follows more closely with the actual readings both temporally and amplitude-wise.

5. Q= 0.0625 and R = 4

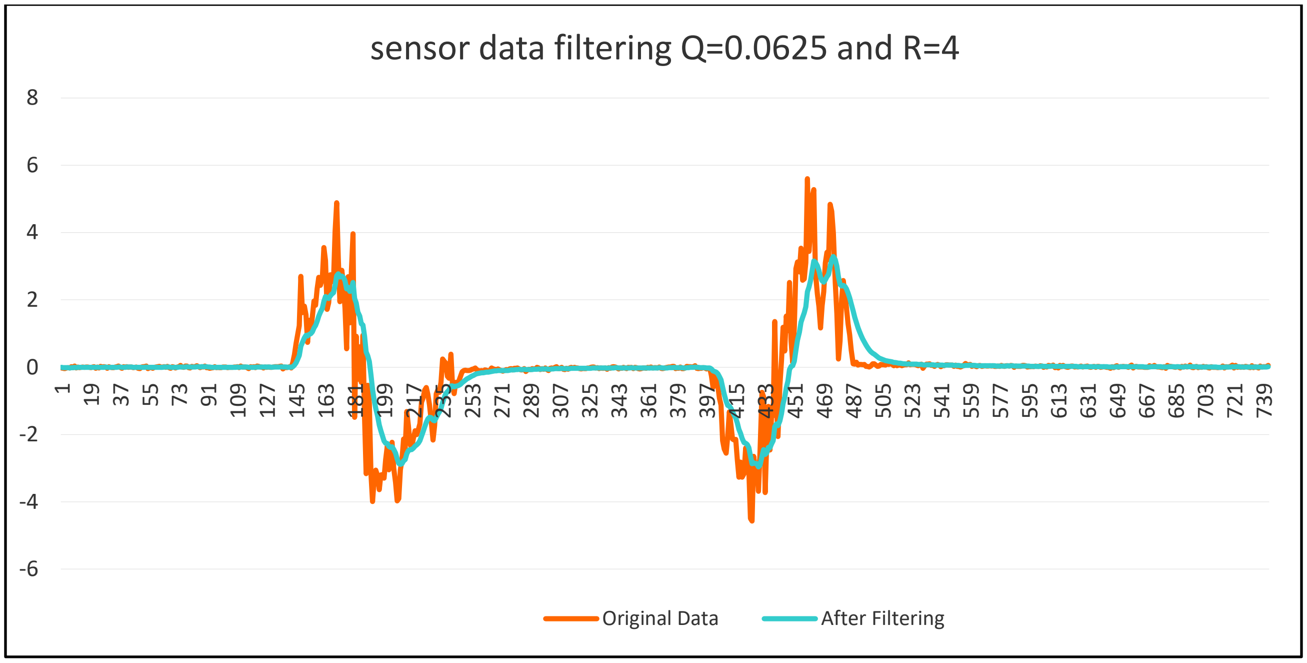


Figure 5. 11 Linear acceleration Sensor Y Axis sensor data filtering with Q=0.0625 and R=4

5.4.3. Filter Linear acceleration through the Kalman Filter

Based on these comparison results, we have decided to use the following settings for the Kalman filter when processing the linear acceleration sensor data in our work: Q = 0.0625 and R = 4, which has shown the best performance and set:

Q matrix = {{0.0625,0,0}, {0,0.0625,0}, {0,0,0.0625};

R matrix = {{4,0,0}, {0,4,0}, {0,0,4}}.

5.4.4. Estimate the Velocity

To estimate the velocity from the linear acceleration data, we have to follow the mathematical integration process. We have developed the following formulas.

Equations:

Velocity data Calculation:

$Velocity_x = Previous\_Velocity_x + Linear\_Acceleration_x * dt;$

$Velocity_y = Previous\_Velocity_y + Linear\_Acceleration_y * dt;$

$Velocity_z = Previous\_Velocity_z + Linear\_Acceleration_z * dt;$

where,

dt = current timestamp - previous timestamp.

Before applying this formula, linear acceleration sensor data must be drift- and sensor noise-free; otherwise, such integration process may amply such drift and noise to distort the results. To reduce this effect, we have used the Kalman filter solution before the integration step. Furthermore, after every integration step, the integration error can again be suppressed using the Kalman filter at the velocity-level. Another potential problem is the initial values of velocity. Here, the initial velocity is unknown to the motion sensors, so we can assume initial velocity as zero, which is reasonable as the object in applications under consideration always starts from a stationary state. In this section, we perform the analysis using the same dataset as shown in Figure 5.6-11.

5.4.4.1. Effects of Sampling rate on Velocity Estimation

As shown in the above formula to calculate velocity, we obtain the *dt* from the timestamps. Timestamp is a value in nanoseconds when the sensor event is generated. So, we will convert timestamp from nanoseconds to seconds, then find the difference between two successive timestamps. We will get velocity in meter per second. The choice of sampling rate is an important factor in integration. Higher sampling rate means small step-size and higher granularity, leading to more accurate integration results. For instance, if the sampling rate is low (for example: - 200 milliseconds or 60 milliseconds), then the sensor data multiply with larger value, resulting in less accurate estimate. Our prototype, e.g., as we discussed earlier, collected sensor samples from Google Nexus 7 every 10 milliseconds. Another important thing is sampling rate may not be consistent, and need to be tracked for each data sample.

5.4.4.2. Velocity Filtering through the Kalman Filter

Obtaining velocity through integration algorithm from linear acceleration may result in a velocity with significant offset and drift away from its actual values [5]. So, the Kalman filter is again used to remove drift and noise in order to minimize integration error. The Kalman filter generates more accurate velocity estimate by suppressing the noise. For that, we pass the unfiltered velocity data (Vx, Vy, Vz) as a measurement vector input into the Kalman filter. Figure 5.12 shows the velocity before and after Kalman filtering.

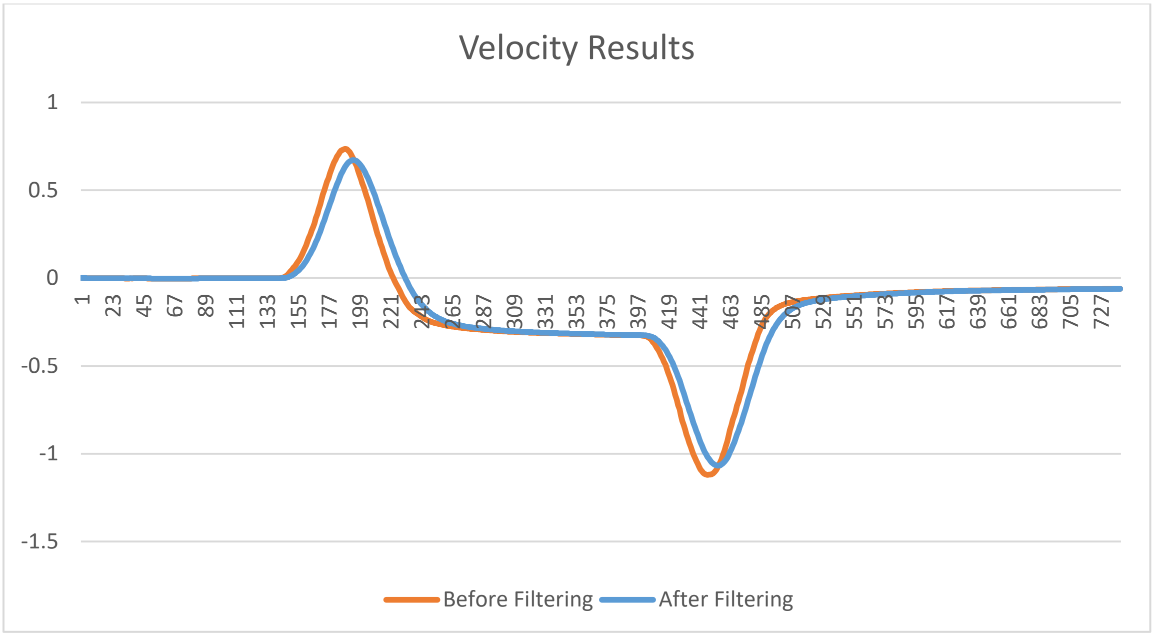


Figure 5. 12 velocity results before and after Kalman Filter

5.4.5. Estimate the Position

To estimate the position from the velocity data, we have to follow another mathematical integration process. We have developed the following formulas.

Equations:

Position data Calculation:

$Position_x = Previous\_Position_x + ((Previous\_velocity_x + Current\_velocity_x)/2) * dt;$

$Position_y = Previous\_Position_y + ((Previous\_velocity_y + Current\_velocity_y)/2) * dt;$

$Position_z = Previous\_Position_z + ((Previous\_velocity_z + Current\_velocity_z)/2) * dt;$

Where, dt = current timestamp - previous timestamp.

Since the initial position is unknown, we assume initial zero position. In real-life application, the initial position can be provided through out-of-band methods, such as GPS for driver-less car, Kinect for body motion tracking, and etc. Similar to calculating the velocity from acceleration, effects of different and varying sampling rate are also considered in this process. Kalman filtered velocity is fed into the algorithms. We will process the same dataset as presented in Figure 5.12 to estimate the corresponding position.

5.4.5.1. Position Filtering through the Kalman Filter

Integration from velocity resulted in position with offset and drift away from the actual values [5]. So, the Kalman filter is again used to remove drift and noise to minimize integration error. The Kalman filter generate more accurate position estimate by suppressing the noise. For that, we pass the unfiltered position data (Px, Py, Pz) as a measurement vector input into the Kalman filter. Figure 5.13 display the estimated position before and after Kalman filtering.

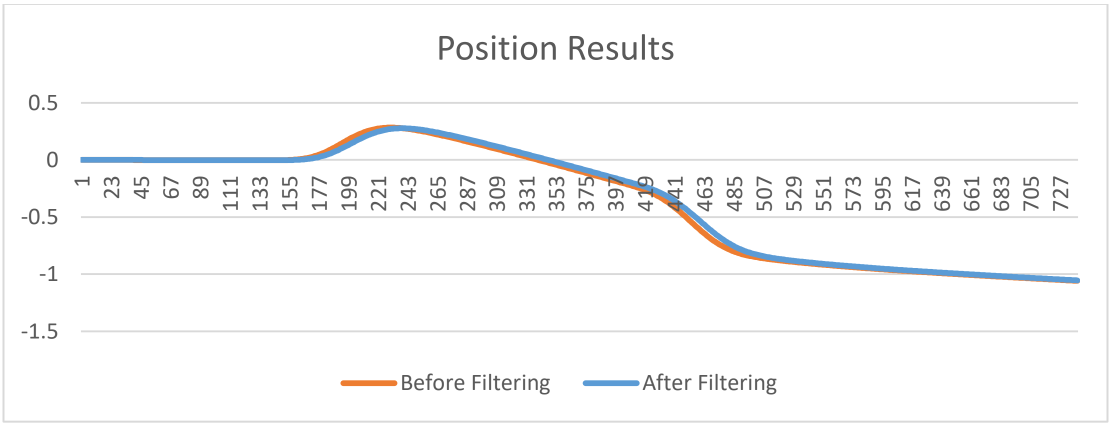


Figure 5. 13 position results before and after Kalman Filter

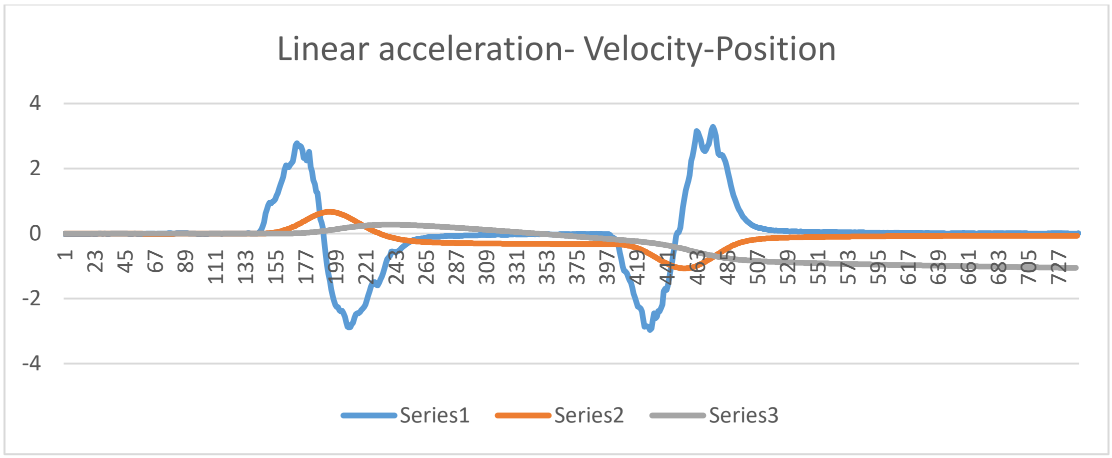


Figure 5. 14 Series1: Linear acceleration, Series2: Velocity, Series3: Position

In summary, Figure 5.14 presents all together the final analysis results for Acceleration, Velocity, and Position Data.

The velocity and, especially, the position results start show the effect of cumulative errors from the integration processes as the subject advances in time and space, indicating the need for additional velocity and/or position data in order to correct the cumulative errors. These side information could be obtained, e.g., through a digital map of the area, either well-established ahead of deployment, or, on the fly by Simultaneous Locating and Mapping based on visual and distance sensors, such as Google-Tango technologies.

5.5. In-Cloud Processing of Sensory Data from the Sony Smartwatch 3 based Sensor Web

In this section, we will present the experiments results of our Sensor-web based Cloud Solution using the completed prototype systems and processing inside the cloud. As discussed in earlier chapters, our sensor-web collects motion data from the Sony Smartwatch 3, and then deliver the data to the cloud database. We will apply the processing algorithms from the preceding sections to analyze sensor data and produce the motion analysis reports. Sony Smartwatch 3 is capable of capturing acceleration data every 4 milliseconds, i.e., sampling rate at 250 samples per second. Figure 5.15 displays the raw acceleration sensor data collected by Sony Smartwatch 3 along the X, Y, Z coordinates.

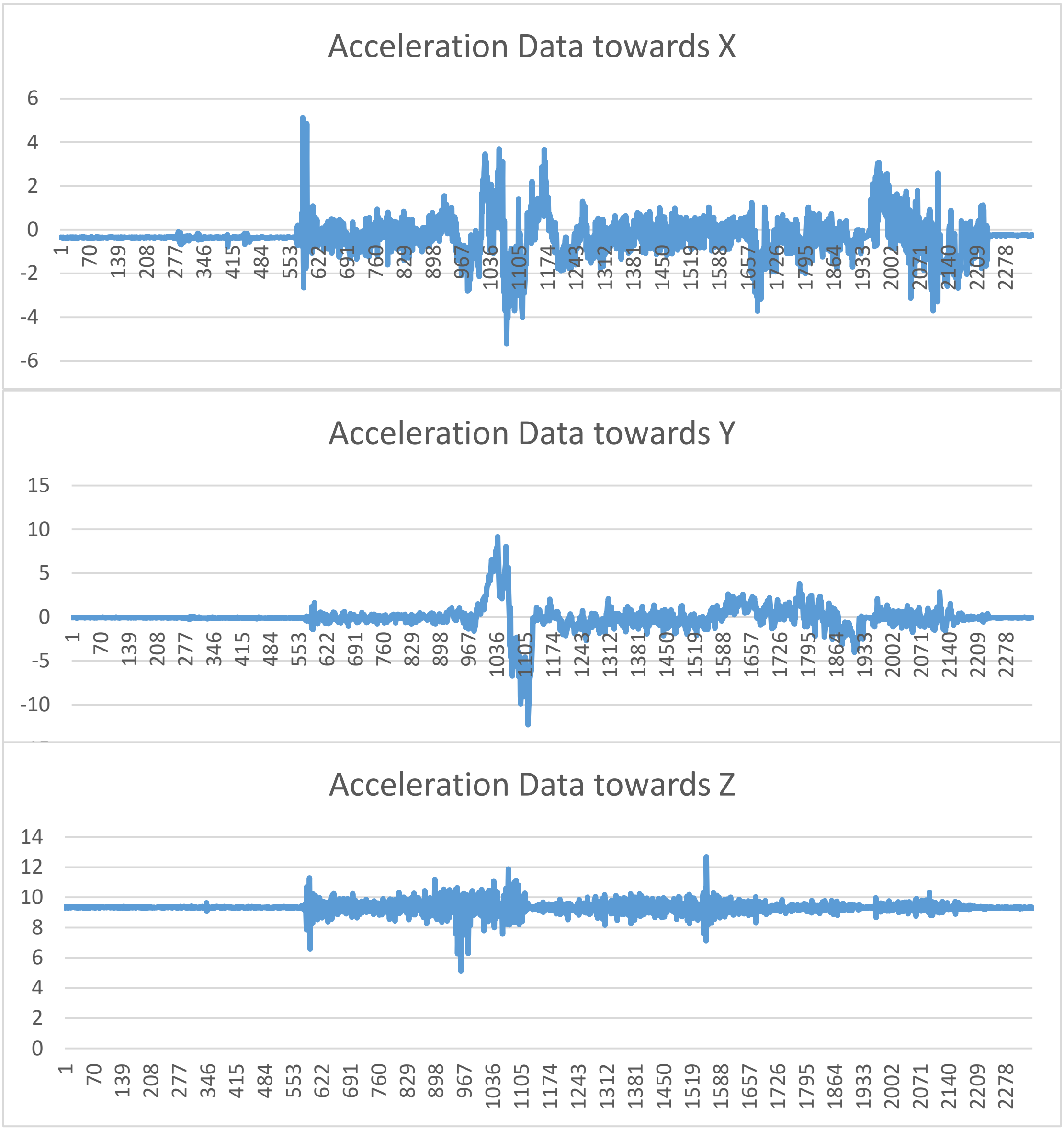


Figure 5. 15 Acceleration Data along the 3 device coordinate axes of Sony Smartwatch 3

Next, we will pass the acceleration Z axis sensor data to our Low-pass/High-pass filter to remove the gravity and generate the Linear acceleration. Fig. 5.16 presents the linear acceleration data produced by using low-pass high-pass filter technique.

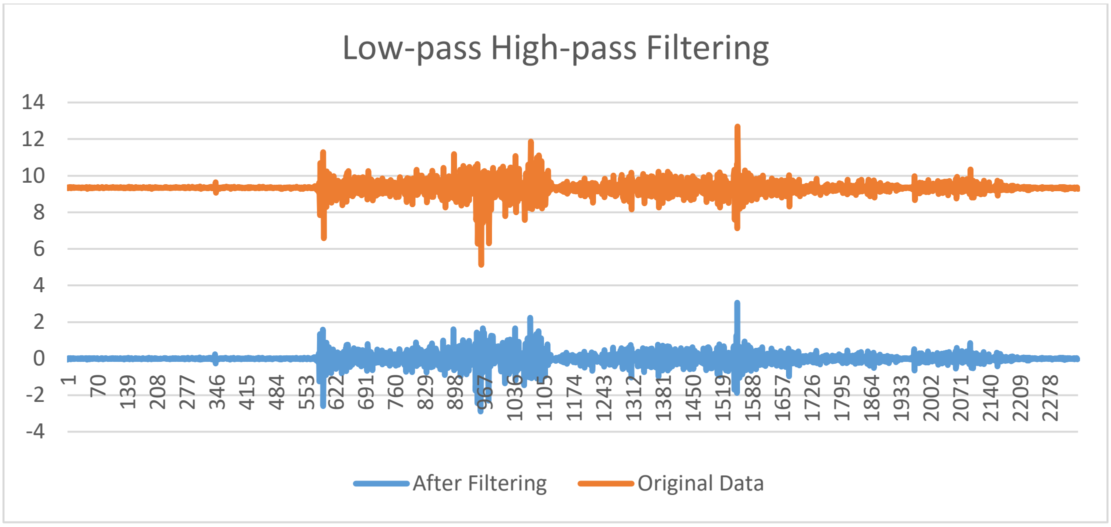


Figure 5. 16 Linear acceleration Data After Low-pass High-pass Filtering

Then, we will analyze these data set to estimate the acceleration, velocity and position as described in the preceding sections. Fig. 5.17-19 shows the estimates produced by our system for acceleration, velocity and position, respectively.

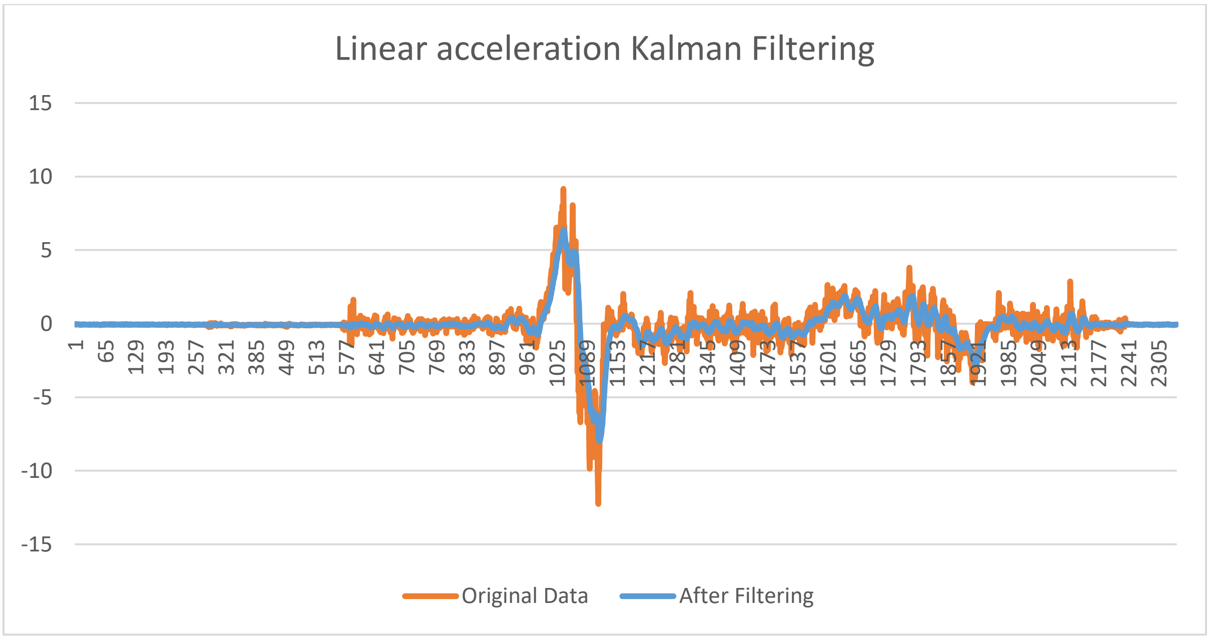


Figure 5. 17 Linear acceleration Kalman Filter Result Analysis

After that, we are using Integration step to calculate Velocity and pass into the Kalman filter to filter out velocity output. Fig. 5.18 describes velocity results before and after Kalman filter step.

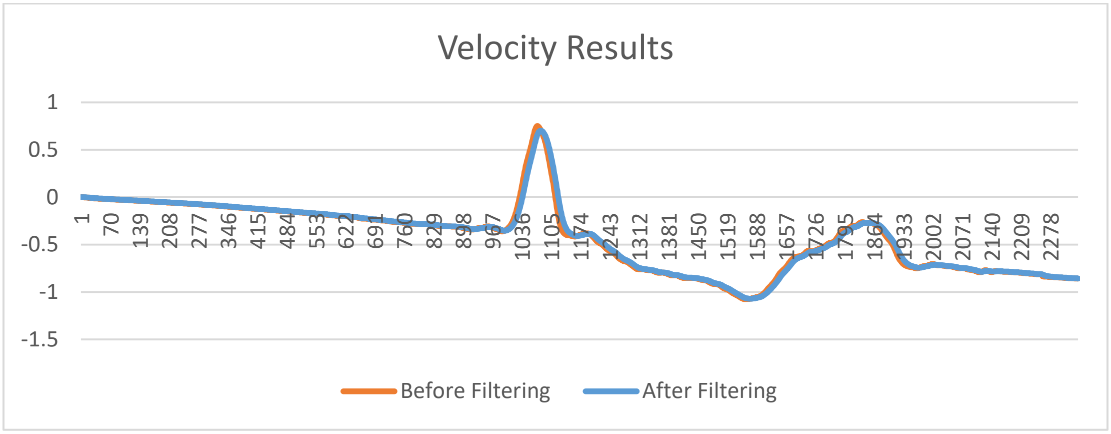


Figure 5. 18 Velocity Output Before and After Kalman Filter

After that, we are using Integration step to calculate position and pass into the Kalman filter to filter out position output. Fig. 5.19 describes position results before and after Kalman filter step.

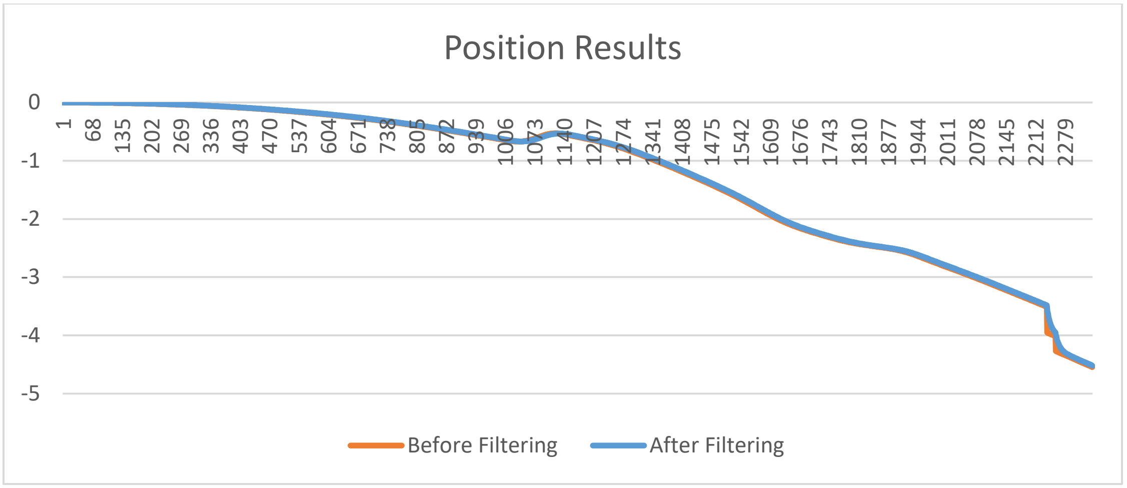


Figure 5. 19 Position Output After Kalman Filter

Again, the velocity and position results also start show the effect of cumulative errors from the integration processes as the subject advances in time and space, indicating the need for additional velocity and/or position data in order to correct the cumulative errors. These side information may be obtained, e.g., through a digital map of the area, either well-established ahead of deployment, or, on the fly by Simultaneous Locating and Mapping based on visual and distance sensors, such as Google-Tango technologies.

Chapter 6

6. Conclusion and Future Work

In this section, we have concluded overall work from all the above chapters as well as describes a lot of new opportunities which can be done in the future using this research work.

6.1. Conclusions

In this thesis, we have concluded the motion tracking and visualization capabilities using the latest motion sensory technologies such as Android Wear Sony smartwatch 3 and google Asus Nexus 7 devices. Android Wearable and handheld devices are low in price and give nice features for development. We analyzed Android Sony smartwatch 3 sensor capabilities, processing capabilities and network capabilities. Also, we have designed sensor web-based cloud solution prototype to collect accelerometer sensor data real-time with higher sampling rate, accuracy, and store huge dataset on a cloud for further processing. We have designed a Stand-alone Motion Capture prototype for Google Nexus 7 device to capture and analyze accelerometer, linear acceleration sensor data with higher sampling rate and store into file for further processing. We have monitored raw noisy sensor data errors and developed low complexity algorithms to support real-time data analysis. Explored different data processing methods to obtain improved-quality motion results from extremely noisy raw data, e.g., by using low pass and high pass filter, and the Kalman filters. We believe this work helps tremendously in Healthcare to identify the causes of lower back pain and assist with the corresponding therapy process. Also, another potential application of this research, is to enable automatic locating and mapping capability inside the GI tract system in order to provide the high-precision navigation capabilities for endoscopy and drug delivery capsules used in GI medicine. We expect that these proposed new technologies will provide improved workflow, accuracy, and efficiency in sports, fitness, and healthcare fields.

Particularly, in this thesis chapter 4, we have designed and developed Sensor-Web based Cloud Solution and Prototype Designs to collect sensor data real-time. We illustrated development capabilities of Android Wear connected with android handheld via Bluetooth using different Android Wear APIs. We concluded accelerometer raw sensor data results from android Sony smartwatch 3. However, linear acceleration sensor is synthetic sensor and not extracting sensor data for Sony smartwatch 3 device case. Also, we have provided cloud based large data storage solution per further processing raw data. Using this prototype, we analyzed higher sampling data rate which is around 4 milliseconds for android watch. A Higher sampling rate is extremely helpful to find accurate velocity and position from acceleration data. We monitored sensor errors present in raw accelerometer sensor data such as noise, drift, offset, gravity, etc. Overall, we provided a cloud based solution to avoid data loss issue, successfully collect sensor data, and store into cloud for further processing.

We have also designed a Stand-alone Motion Capture prototype to collect sensor data from google Nexus 7. The reason behind this is the google Nexus 7 linear acceleration provides good results. So, we are using this data in Kalman filter based low complexity algorithm to estimate acceleration, velocity, and position. We have monitored 10 milliseconds sampling data rate for google Nexus 7 acceleration, linear acceleration sensor data extraction. We are storing data into file for further processing. We have analyzed sensor errors present in raw linear acceleration sensor data such as noise, drift, offset, gravity etc.

Specifically, we have developed low complexity based algorithm to remove sensor errors from raw sensor data and estimate motion results such as velocity and position from acceleration. We discussed raw sensor data and integration errors for raw linear acceleration sensor data samples. We used a low pass filter to identify gravity and a high pass filter to produce linear acceleration from the raw acceleration data. We provided Kalman Filter based De-noising and Motion Data Estimation algorithm for step by step processing to estimate acceleration, velocity and position. We discussed results of this algorithm for both Android Wear and google Nexus 7 datasets. Following are advantages and limitations of our low complexity estimation algorithm:

Advantages:

1. Remove constant offset from Linear acceleration raw sensor data.
2. Filter sensor and process noise through the Kalman filter implementation.
3. Minimize integration error and calculate accurate acceleration, velocity and position sensor data.
4. Using Higher and consistent sampling rate deduct sensor errors while integration.
5. Estimate the current location of smartwatch or linear acceleration sensor data.
6. Correct Microsoft Kinect Human body movement position data to remove unused data to make movement more adjustable and accurate. Since it requires

small amount of memory we can transfer data through online without any kind of delay.

Limitations:

1. Challenge to find a right value for process noise Q and sensor noise R and also difficulty to set other parameters of the Kalman filter.

In summary, we have designed and developed prototypes to analyze motion tracking capabilities of Android Wear and google Nexus 7. After that analyze sensor errors and develop low complexity based filter techniques algorithms to analyze motion results. These technologies are extremely helpful in healthcare, sports and fitness fields.

6.2. Future Work

In the future work, one could combine sensor-web based cloud solution prototype and the Kalman filter based prototype together. We are extracting a sensor data real-time, but in future, we will use Kalman filter and lowpass-highpass filter techniques for data processing in android handheld platform, which can produce motion results real-time and display on android handheld application.

On the motion results, velocity and position results become not as accurate as subject advances in time and space, we could use dead reckoning technique to make velocity and position more accurate by utilize additional velocity and position data made available through other means. Dead reckoning is the process of calculating one's current position by using a previously determined position, or fix, and advancing that position based upon known or estimated speeds over elapsed time and course.

We believe these future works will help mature this technology for the real-world sport, fitness and healthcare fields.

7. References

[1] xsens.com, “The idea behind Valedo” [Online].

Available:https://www.xsens.com/customer-cases/valedo-back-pain-therapy-xsens-technology/

[2] invensense.com, “Overview,” 9 Axis Motion Tracking. [Online]

Available: https://www.invensense.com/products/motion-tracking/9-axis/

[3] Hol, J. (2011). Sensor fusion and calibration of inertial sensors, vision, ultra-wideband and GPS (Unpublished doctoral dissertation). Diss. Linköping: Linköpings universitet.

[4] Milette, G., & Stroud, A. (2012). Professional Android sensor programming. Indianapolis, IN: Wiley.

[5] Slifka, L. (2004). AN ACCELEROMETER BASED APPROACH TO MEASURING DISPLACEMENT OF A VEHICLE BODY

[6] sonymobile.com. “Sony White paper on SWR50 Smartwatch 3” [Online]. Available: http://www-support-downloads.sonymobile.com/swr50/whitepaper_EN_swr50_smartwatch3_2.pdf

[7] ______, “Sensor API Overview” [Online]

Available: https://developer.android.com/guide/topics/sensors/sensors_overview.html

[8] ______, “Sending and Syncing Data” [Online]

Available: https://developer.android.com/training/wearables/data-layer/index.html

[9] ______, “Debugging over Bluetooth” [Online]

Available: https://developer.android.com/training/wearables/apps/bt-debugging.html

[10] _____, “DataItem API Overview” [Online]

Available: https://developer.android.com/training/wearables/data-layer/data-items.html

[11] _____, “Message API Overview” [Online]

Available: https://developer.android.com/training/wearables/data-layer/messages.html

[12] _____, “Filter for Motion Sensors” [Online] Available: https://developer.android.com/guide/topics/sensors/sensors_motion.html#sensors-motion-accel

[13] richard@UTCS, “Kalman Filter Simulation” [Online]

Available: https://www.cs.utexas.edu/~teammco/misc/Kalman_filter/

[14] Interactive Matter Lab, “Filtering Sensor Data with a Kalman Filter”

Available:http://interactive-matter.eu/blog/2009/12/18/filtering-sensor-data-with-a-Kalman-filter/

[15] Wikipedia, the free encyclopedia, “Android Wear” [Online]

Available: https://en.wikipedia.org/wiki/Android_Wear

[16] Megha Dattatrey Dalvi, (Jan. 2017). Customizable 3-D Virtual GI Tract Systems for Locating, Mapping and Navigation inside Human Gastrointestinal Tract