Media Engineering and Technology Faculty
German University in Cairo

Sensor Fusion for Position Tracking

Bachelor Thesis

Author: Aya Khaled Youssef Sayed
Supervisor: Dr. Mohamed Ahmed Elmahdy
Submission Date: 15 May, 2017

This is to certify that:

(i) the thesis comprises only my original work toward the Bachelor Degree

(ii) due acknowlegement has been made in the text to all other material used

---

Aya Khaled Youssef Sayed
15 May, 2017

Acknowledgments

First and foremost, I thank Allah for all the countless blessings, for giving me the strength and perseverance to complete this project till the very end, and for providing me with the knowledge needed to be able to work on it.

I would like to express my deep gratitude to all who supported me through my thesis and made it happen. I sincerely thank my supervisor, Dr. Mohamed Elmahdy, for the continuous guidance, assistance and help he offered me all the way. He was not only providing me with the material I needed, but also he was always there whenever I needed him and showed a great deal of motivation. The knowledge and expertise he generously shared with me were invaluable to the completion of this thesis.

Furthermore, I extend my deepest appreciation to my parents for their unconditional love and support. A special thank you goes to my sisters for their genuine love and their joyful presence. In fact, my family are the real reason behind my success.

Last but not least, I wholeheartedly thank all my friends for their encouragement and for making my college years as fun and enjoyable as they were.

Abstract

This project examines the level of accuracy that can be achieved in position tracking by using built-in sensors in an Android smartphone as well as external Arduino hardware sensors. It is focused on improving translation and orientation accuracy besides detecting the movement direction of the cellular phone. The approach is based on sensor fusion using data from the device's different built-in sensors like accelerometer, gyroscope, and magnetometer. This is because the output from individual sensors is noisy and inaccurate, hence, it needs to be improved and smoothed to get rid of the error due to vibration in accelerometers and drift in gyroscopes. Kalman and Complementary filters are examples of the implemented sensor fusion methods while median filter and average method are examples of the smoothing techniques applied in the Arduino project and Android application as well. The results show that the noise is significantly filtered out and graph spikes are eliminated when smoothing techniques are applied. The accuracy percentage of sensor fusion algorithms have outperformed the accuracy of individual sensors. Moreover, these enhancement techniques are promising for future handheld indoor localization and navigation systems that can be used in numerous places such as malls, museums, tunnels, large office buildings and hospitals. The project could be helpful later on for biomedical in addition to entertainment applications like interactive games.

Contents

Chapter 1

Introduction

This project is an embedded system project. An embedded system is an electronic system that uses a computer chip and contains at least one controlling device but in such a way that it is hidden from the end user. Such systems use microcontrollers (MCUs) or microprocessors (MPUs), or they may use custom-designed chips. Embedded systems are employed in automobiles, planes, trains, space vehicles, machine tools, cameras, consumer electronics, office appliances, network appliances, video games, cellphones, GPS navigation, as well as robots and toys. Low-cost consumer products can use microcontroller chips that cost less than a dollar.

1.1 Motivation

The majority of people always think about the camera resolution for example when it comes to the idea of mobile phone features and no one care about the types and the quality of the sensors in their mobile phone. In spite of the fact that the mobile phone sensors are deprecated and are not widely used, they are quite powerful and crucial in our daily life. For example, the accelerometer in smartphones is used for determining the orientation of the mobile and rotate their display between portrait and landscape mode depending on the phone tilt although it can be used in indoor positioning and motion direction detection when the Global Positioning System (GPS) signal is weak or unavailable. Furthermore, GPS does not provide accurate information in closed areas because walls impede the signals sent from the satellites orbiting Earth, however, the gyroscope can provide accurate directions in this case. On the other hand, the quality of the sensor individually is poor and there is a significant amount of noise. Therefore, it needs to be enhanced and fused with other sensors in order to obtain more accurate results.

1.2 Aim of the Project

The main objective of this project is to enhance the Android sensors in smartphones as well as the Arduino sensor MPU-6050 and fuse multiple sensors data in order to obtain accurate orientation and motion direction. The project consists of two parts; Android mobile application to access the mobile phone sensors and the second part is implemented using Arduino and Processing IDE in order to communicate with the hardware sensor. Several multisensor fusion algorithms are implemented in both applications.

1.3 Thesis Outline

The thesis is divided into 5 chapters, where this chapter covered the introduction. Chapter 2 discusses the background needed to understand the constituents of the project. Moreover, the sensors used and their specifications, as well as the theoretical part of the implemented sensor fusion and smoothing algorithms and any other needed techniques are explained. This chapter is not only providing an overview of the platform and hardware that are used in this project, but also other related and similar work. Chapter 3 demonstrates the Android application and the Arduino project design and workflow. The rotation and translation modes in addition to the implementation process of the enhancement techniques and smoothing methods are also covered in detail in this chapter. The experimentation, results and algorithms comparison are provided in Chapter 4. Finally, Chapter 5 summarizes all the work presented in this thesis and suggests extra features and more enhancement ways that can be added in the future. At the end of the thesis, an appendix can be found which contains a list of all the abbreviations used throughout, as well as a list of all of the screenshots and figures.

Chapter 2

Background

2.1 Sensors in Android

The sensors supported by android devices are the ones used in the first part of this project. The Android sensor framework lets you access many types of sensors. In fact, Hardware and Software sensors are the two basic types of sensors in Android. Hardware sensors are physical components that are built into the device to measure specific environmental properties such as acceleration, angular change or geomagnetic field strength. Accelerometer (*TYPE_ACCELEROMETER*), gyroscope (*TYPE_GYROSCOPE*), and magnetometer (*TYPE_MAGNETIC_FIELD*) are examples of hardware sensors. Software sensors are known as virtual or synthetic sensors. These sensors derive their data from hardware-based sensors such as *TYPE_ORIENTATION* and *TYPE_ROTATION_VECTOR* [1]. This project is mainly based on hardware sensors and the data from these sensors are used to monitor the motion direction of a device and can be fused to track orientation. The values returned by android sensors are expressed in a specific static frame or coordinate system that is relative to the device's screen in its default orientation. The axes are not swapped even if the orientation of the device changes such that the x-axis is always horizontal and the positive direction points to the right. The y-axis positive direction is vertical and points upwards while the z-axis points outside the front face of the mobile phone screen as shown in Figure 2.1.

2.1.1 Accelerometer

Cellphone accelerometers are based on tiny microchips with all their components chemically etched onto the surface of a piece of silicon. An accelerometer is an electromechanical device used to measure acceleration forces. Such forces may be static, like the continuous force of gravity or, as is the case with many mobile devices, dynamic to sense movement or vibrations. There are two types of accelerometers supported by android. The first one is called *TYPE_ACCELEROMETER* and it measures the acceleration in the X, Y, and Z axes including the gravitational force. The other one is

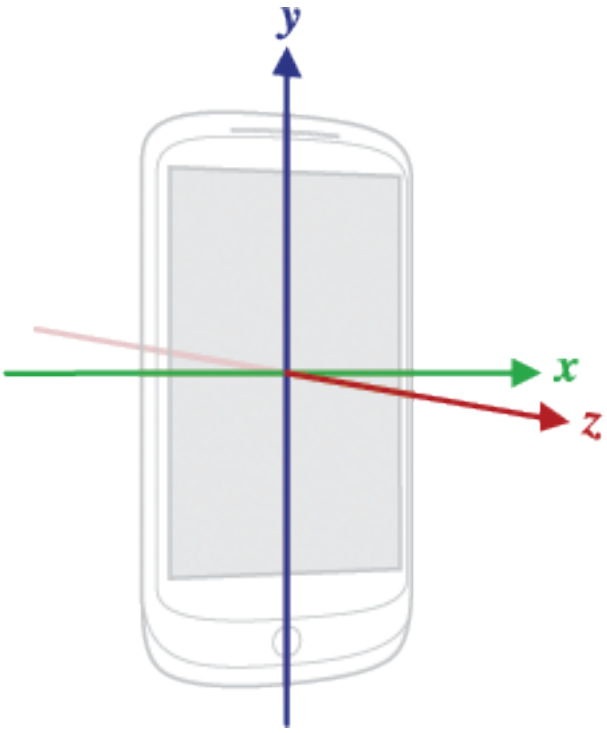

Figure 2.1: Coordinate system (relative to a mobile device) used by the Sensor API

called *TYPE_LINEAR_ACCELERATION* and it measures the real acceleration of the device in the three dimensions excluding the contribution of the force of gravity from the accelerometer data. Theoretically, linear acceleration is the result of subtracting the gravity vector (the vector pointing towards the center of the Earth) from the acceleration. Practically, linear acceleration can be obtained by applying a high-pass filter on the acceleration to eliminate gravitational forces and reduce noise [3]. The measured acceleration in both types is in meters/$second^2$ and is relative to the Earth. Generally, accelerometers are extremely sensitive and have a good absolute reading at low frequency. However, they became noisy and inaccurate to an extent that they cannot distinguish the acceleration due to gravity as the frequency increases. The most common application of accelerometers in consumer electronics is switching between portrait or landscape display modes.

2.1.2 Gyroscope

The main functionality of the gyroscope is to measure the rate of rotation for the three coordinate axes in radians/second. An important note to be made is that the measured angular velocity is relative to the body. There are two types of gyroscopes in android; the calibrated gyroscope with drift eliminated and the uncalibrated one that returns raw values from the sensor with drift. Drift is the small errors that are introduced each iteration and they are added up over time. The *TYPE_GYROSCOPE* returns only the values of angular speed in the X, Y, and Z directions while the *TYPE_GYROSCOPE_UNCALIBRATED*

reports the estimated drift on each axis as well and no gyro drift compensation is performed [3]. This drift is represented as an offset value returned by the sensor when it is motionless, so the measurement will not return to zero when the object is moved back to its original position. The values returned by the calibrated gyroscope are calculated as the values of each axis from the uncalibrated gyroscope minus the returned expected drift values of the same axis. Gyroscope data is reliable only in the short term as it starts to drift on the long term.

2.1.3 Magnetometer

The magnetometer mainly acts as a compass and measures the ambient geomagnetic field strength that is relative to the Earth for all three physical axes (X, Y, and Z) in micro Tesla (μT). It is used to determine the orientation of the device like the gyroscope. It calculates the magnetic azimuth which is the angular distance from the magnetic north. It is quite sensitive to the environmental magnetic disturbances when measuring direction, unlike the gyroscope.

2.1.4 Rotation Vector

Rotation vector, known as *TYPE_ROTATION_VECTOR*, is a virtual sensor in Android that represents the orientation of the device relative to the East-North-Up coordinates frame as a combination of an angle and an axis. The East-North-Up coordinate system is defined as a direct orthonormal basis where X and Y point east and north respectively and are tangential to the ground while Z points towards the sky and is perpendicular to the ground. The orientation is usually obtained by integration of accelerometer, gyroscope, and magnetometer readings. Then, some smart calculations are done to provide more accurate information rather than using raw data from hardware sensors. One of the advantages of this sensor is that it does not suffer from the gimbal lock problem since the orientation is reported using quaternion coordinates. An example for gimbal lock is one of the coordinates goes to 90 or -90 degrees when the actual orientation vector is close to vertical and the remaining two coordinates become either uninterpretable or dangerously denormalized. The main aim of using this sensor in the project is to compare its performance with the implemented sensor fusion techniques which use the same hardware sensors.

2.2 Arduino Sensor MPU-6050

The other type of sensors used in this project is the MPU-6050 sensor as illustrated in Figure 2.2. The MPU-6050 is a 6 Degrees of Freedom (DOF) or a six-axis IMU sensor, which means that it gives six values as output. This sensor is connected to an Arduino

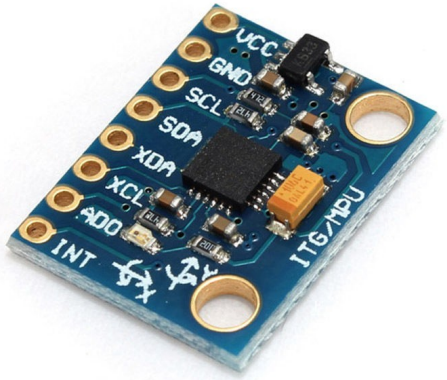

Figure 2.2: MPU-6050 accelerometer/gyroscope sensor

Uno board to read acceleration and angular velocity using Inter-Integrated Circuit (I2C) communication protocol. The MPU-6050 always acts as a slave to the Arduino with the Serial Data (SDA) and Serial Clock (SCL) pins connected to the I2C-bus. Arduino IDE is used to get the raw data from the main microcontroller unit in this project. Then, the received measurements are sent serially to processing program for further data processing. This sensor contains a three-axis accelerometer and a three-axis gyroscope in a single chip, so they do not need to be aligned. It is very accurate since it contains 16-bits analog to digital conversion hardware for each channel. Therefore, it captures the X, Y, and Z channels at the same time [14]. Furthermore, it is absolutely great for DIY projects. The MPU-6050 sensor specifications are shown below in Table 2.1.

2.2.1 Accelerometer Features

According to the datasheet of MPU-6050 [14], the raw output analog signal obtained from the triple-axis accelerometer is actually the force acting on the body in each of the three axes represented in terms of g. And hence, some calculations are needed to translate these measurements into degrees in the case of calculating yaw, pitch, and roll or into meters in case of calculating the distance by double integration. First of all, a full-scale range has to be selected like ±2g, ±4g, ±8g or ±16g. Then, the output of the accelerometer is multiplied by 9.81 and passed over a high-pass filter to remove the gravitational force contribution and any other noisy signals. The next step is to integrate the output signal over time to get the velocity. Finally, the integration of velocity over time will produce the displacement which is the desired output. However, a significant amount of error is noticed after the double integration because it is accumulated over time. There are many different types of accelerometers. The mechanical ones which have something like a mass attached to a spring suspended inside an outer casing. The casing moves

MPU-6050 Specifications	
Model	GY-521
Material	PCB + Plastic + Copper
Power Supply	3 5V
Communication Mode	Standard IIC communication protocol
Features	- Chip built-in 16-bit AD converter, 16-bit data output. - Immersion Gold plating PCB, machine welding process to ensure quality.
Gyroscope Range	±250, ±500, ±1000, ±2000 °/sec
Accelerometer Range	±2, ±4, ±8, ±16 g
Accuracy	Accelerometer: 220 g/$\sqrt{Hz}$, Gyroscope: 0.03 °/s/$\sqrt{Hz}$
Pin pitch	2.54mm
Dimensions	0.83 in x 0.63 in x 0.12 in (2.1 cm x 1.6 cm x 0.3 cm)
Weight	0.18 oz (5 g)

Table 2.1: Sensor MPU-6050 specification table

during acceleration while the mass lags behind causing the spring to stretch by a distance that is proportional to the stretching force due to acceleration. An alternative type is the piezoresistive accelerometers that are based on generating electrical or magnetic signals according to the size of the force acting on it. Capacitors can also be used in accelerometers such that the acting force is measured by the capacitance change. Last but not least, the MPU-6050 accelerometer output could be used directly to calculate the yaw, pitch, and roll. Roll is the rotation around a front-to-back axis (y-axis), yaw is simply rotating left and right around a vertical axis (z-axis) and pitch is rotating up and down around a side-to-side axis (x-axis) as shown in Figure 2.3. Nevertheless, yaw can not be determined using the accelerometer in MPU-6050 sensor because gravity is not acting on that axis. The general formulas that are used to calculate pitch, roll, and yaw from accelerometer data are:

$$pitch = \tan^{-1}\left(\frac{accelerationX}{\sqrt{accelerationY^2 + accelerationZ^2}}\right) \times \frac{180}{PI} \tag{2.1}$$

$$roll = \tan^{-1}\left(\frac{accelerationY}{\sqrt{accelerationX^2 + accelerationZ^2}}\right) \times \frac{180}{PI} \tag{2.2}$$

$$yaw = \tan^{-1}\left(\frac{accelerationZ}{\sqrt{accelerationX^2 + accelerationZ^2}}\right) \times \frac{180}{PI} \tag{2.3}$$

2.2.2 Gyroscope Features

The MPU-6050 Micro Electro-Mechanical System (MEMS) gyroscope is tiny and contains vibrating elements that measure the Coriolis effects. That is the gyroscope rotation causes

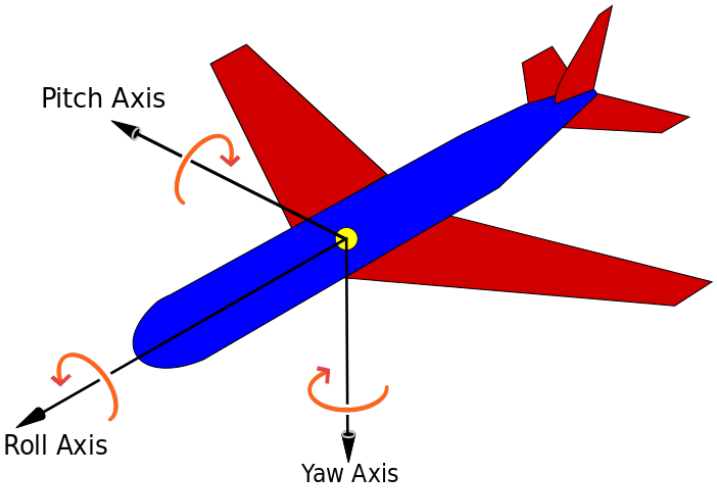

Figure 2.3: Roll, pitch, and yaw

a small resonating mass to vibrate along a drive axis and a secondary vibration is induced along the perpendicular sense axis due to the Coriolis force. In order to calculate the angular velocity, this secondary rotation is measured and converted into very low-current electrical signals that can be amplified and read by a host microcontroller. The gyroscope in MPU-6050 sensor supports a wide range of degrees per second that can be selected to detect the degrees that can be rotated in one second. For example, if the selected range is equal to ±250 °/sec, it means that only $\frac{250}{360} \times 60 = 41.67$ revolutions per minute (rpm) can be detected. Then, according to the selected range, the values measured by the gyroscope have to be divided by the gyro sensitivity value as mentioned in the datasheet [14] in order to obtain the angular velocity for each axis. For example, if the selected gyro full-scale range is ±250 °/sec, the gyro sensitivity value will be 131 LSB/°/sec. In order to get the angle of rotation, the angular velocity is integrated with time for each axis. The rotation angle increases in the anti-clockwise direction and decreases when the device is rotated in the clockwise direction.

2.3 Sensor Fusion

Sensor fusion is combining the result of several sensors such as the accelerometer, magnetometer, and gyroscope to track the position when the GPS signal is lost in the tunnel for example [18]. It upgrades the motion sensing capabilities such that the overall result is better than the output when each sensor is used individually [19]. This is because the sensor measurements are noisy and taken at discrete points in time, so multiple measurements are fused to reconstruct the parameter of interest. In addition, there are small errors that are introduced in each sensor reading and these errors are accumulated over time resulting in a nonsense measurements. Hence, sensor fusion techniques such as complementary filter can overcome this problem by improving the signal's noise caused by

the accelerometer and magnetometer sensors and the gyro drift as well. Sensor fusion is used in numerous applications like robotics, navigation, augmented reality, industrial controls and human body position monitoring which is widely used in gaming, military, and some medical applications. Generally, sensor fusion is a number of algorithms and methods. Kalman filter, Complementary filter, Central limit theorem, Bayesian networks, and Dempster-Shafer are examples of sensor fusion techniques.

2.3.1 Kalman Filter

The method was developed by Kalman and Bucy in 1960 [19]. It is also known as Linear Quadratic Estimation (LQE). Kalman filter is mainly based on trying to estimate the state of the system, based on the current and previous states, that tend to be more precise that than the measurements alone. The algorithm uses a series of measurements observed over time and it is divided into two phases; prediction and update. In the prediction step, Kalman filter estimates the internal state of a linear dynamic system given a sequence of noisy observations. In the next step, these estimated current state variables with their uncertainties are updated such that more weight will be given to estimates with higher certainty when the output of the next measurement including error and noise is determined. In fact, there are two types of noise in this case. The first one is the noise of the input to the filter which is called the measurement noise while the other type is the noise of the system itself and it is called the process noise. This algorithm is recursive and optimal for linear systems and requires only the current input measurements and the previously calculated state and its uncertainty matrix to calculate the new state of the system instead of using the whole history. In addition, it is a widely applied concept in time series analysis and well-suited for real-time applications. Extended Kalman filter and unscented Kalman filter have been developed as an extension to improve the standard Kalman filter method as they work on nonlinear systems.

The System State x_k

The Kalman filter model assumes the true state at time k is evolved from the state at (k - 1) according to the following equation:

$$x_k = F_k x_{k-1} + B_k u_k + w_k \tag{2.4}$$

where F_k is the state transition model which is applied to the previous state x_{k-1}, B_k is the control-input model which is applied to the control vector u_k, w_k is the process noise which is Gaussian distributed with a zero mean and with covariance Q to the time k, and x_k is the state matrix which is given by:

$$x_k = \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_k$$

Therefore, the output of the filter will be the angle θ and the bias $\dot{\theta}_b$ based upon the measurements from the accelerometer and gyroscope as well. The bias is the amount the gyro has drifted. This means that one can get the true rate by subtracting the bias from the gyro measurement [17] [15]. Moreover, F_k, B_k, and w_k are defined as:

$$F_k = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix}$$

$$B_k = \begin{bmatrix} \Delta t \\ 0 \end{bmatrix}$$

$$w_k \sim N\left(0, Q_k\right)$$

Q_k is the process noise covariance matrix and is given as:

$$Q_k = \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{bmatrix} \Delta t$$

The measurement z_k

At time k an observation (or measurement) z_k of the true state x_k is made according to equation:

$$z_k = H_k x_k + v_k \tag{2.5}$$

where H_k is the observation model which maps the true state space into the observed space and v_k is the measurement noise which have to be Gaussian distributed as well with a zero mean and R as the covariance [17] [15]. H_k and v_k are given by:

$$H_k = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$v_k \sim N\left(0, R\right)$$

Kalman Filter Equations

Kalman Filter can be implemented using 7 equations; 2 for the prediction part and the other 5 are responsible for the update part. Equations 2.6 and 2.7 below are demonstrating the prediction part of the algorithm. The first equation predicts the current state based on all the previous states and the gyro measurement while the second equation estimates the error covariance matrix at time k.

$$\hat{x}_{k|k-1} = F\hat{x}_{k-1|k-1} + B\dot{\theta}_k \tag{2.6}$$

where $\hat{x}_{k|k-1}$ is called a priori state. A priori means the estimate of the state matrix at the current time k based on the previous state of the system and the estimates of the states before it.

$$P_{k|k-1} = FP_{k-1|k-1}F^T + Q_k \tag{2.7}$$

where $P_{k|k-1}$ is called a priori error covariance matrix and $P_{k-1|k-1}$ is the previous error covariance matrix. The error covariance matrix P is a 2×2 matrix in this case. In the update part, the difference between the measurement z_k and the priori state $x_{k|k-1}$ is computed as shown in Equation 2.8.

$$\tilde{y}_k = z_k - H\hat{x}_{k|k-1} \tag{2.8}$$

where y_k is called the innovation and it is not a matrix and H is the observation model which is used to map the a priori state $x_{k|k-1}$ into the observed space (the measurement from the accelerometer). In the next step, the innovation covariance is calculated using Equation 2.9. The purpose of this equation is to estimate how much the measurement based on the priori error covariance matrix $P_{k|k-1}$ and the measurement covariance matrix R should be trusted. The observation model H is used to map the priori error covariance matrix $P_{k|k-1}$ into observed space. Notice that S is not a matrix and it is directly proportional to the measurement noise value. The larger the S value, the less reliable the incoming measurement will be.

$$S_k = HP_{k|k-1}H^T + R \tag{2.9}$$

Then, the Kalman gain which is used to to indicate how much we trust the innovation is calculated using Equation 2.10. In this equation, the transpose of the observation model H is used to map the state of the error covariance matrix P into observed space. The

error covariance matrix is then multiplied by the inverse of the innovation covariance S. The result Kalman gain is a 2×1 matrix.

$$K_k = P_{k|k-1} H^T S_k^{-1} \tag{2.10}$$

Finally, Equations 2.11 and 2.12 are responsible for updating the posteriori estimate of the current state and error covariance matrix respectively. In Equation 2.11, the priori state $\hat{x}_{k|k-1}$ is added to the Kalman gain and then multiplied by the innovation $\tilde{y}_k$. Notice that the innovation could be positive or negative since it is the difference between the measurement z_k and the estimated priori state $\hat{x}_{k|k-1}$. In a nutshell, Kalman filter is basically self-correcting the error covariance matrix based on how much the estimate is corrected [15].

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \, \tilde{y}_k \tag{2.11}$$

$$P_{k|k} = (I - K_k H) P_{k|k-1} \tag{2.12}$$

where I is called the identity matrix and is defined as:

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

2.3.2 Complementary Filter

This filter can be used to merge the signals returned from the sensors and mix the accelerometer with the gyroscope as a single sensor cannot be used individually since the accelerometer has much noise due to vibrations and the gyroscope suffers drift. This causes wrong data values returned by the sensors at the end, thus Complementary filters are used because they manage both high-pass and low-pass filter simultaneously and combine the strengths of each sensor to obtain more accurate signal and useful result. This technique depends on filtering acceleration from vibration by using the low-pass filters which filter out high-frequency signals. Moreover, it uses high-pass filters to attenuate low-frequency signals such as gyro drift and then combine both results. It is great for many applications and not complicated like Kalman filter which requires a lot of mathematical background in Euler angles and quaternions and needs too many calculations such as the coefficients of the matrices, the process-based error, measurement error. On the contrary, Kalman filter is more efficient and Complementary filter has limitations in certain tasks. Furthermore, Complementary filter have different orders. The second order Complementary filter is more complicated than the first order. On the other hand, the second order is more accurate and efficient [4].

First Order Complementary Filter

The first order Complementary filer is simple and can be described by a single equation:

$$\theta_{t+1} = k(\theta_t + \omega\Delta t) + (1 - k)\phi \tag{2.13}$$

where $\omega\Delta t$ is the change in angle over a certain amount of time. The first part of the equation $(\theta_t + \omega\Delta t)$ is a calculation of the current angle if only the gyroscope was being used while the other part (ϕ) is the calculation of the current angle if only the accelerometer was being used. The weighted constant (k) is generally a value chosen after experimental testing that is part of the testing process.

Second Order Complementary Filter

The principle of the second order Complementary filter is the same as the first order, but the algorithm is more complicated and the results are better. Figure 2.4 shows the structure of the second order filter. It can be described by the following equations:

$$X_1 = (accel\_angle - \theta_t) \times k^2 \tag{2.14}$$

$$Y_{t+1} = dt \times X_1 + Y_t \tag{2.15}$$

$$X_2 = Y_{t+1} + (accel\_angle - \theta_t) \times 2 \times k + gyro\_rate \tag{2.16}$$

$$\theta_{t+1} = X_2 \times dt + \theta_t \tag{2.17}$$

where θ_{t+1} is the current result angle from second order Complementary filter and θ_t is the previous result. accel_angle is the calculated angle using the accelerometer and it could be yaw, pitch or roll according to the axis around which the rotation angle is calculated. gyro_rate is the angular velocity around the same axis measured from the gyroscope. dt is the loop time and k is a constant [4].

2.3.3 High-pass Filter

The high-pass filter generally allows signals with a frequency higher than a cutoff frequency to pass and attenuates the signal that is steady over time [11]. It is mainly used to eliminate the gravity offset and get the real acceleration of a moving body from accelerometer data. This is because when the device is motionless with no acceleration, an acceleration with magnitude of 9.81 m/s^2 is measured by the accelerometer. However, when the device is free falling meaning that it is accelerating towards the ground at 9.81 m/s^2, the accelerometer reads a magnitude of 0 m/s^2. Hence, high-pass filter is needed to remove the gravitational force contribution from the acceleration and reduce noise [18].

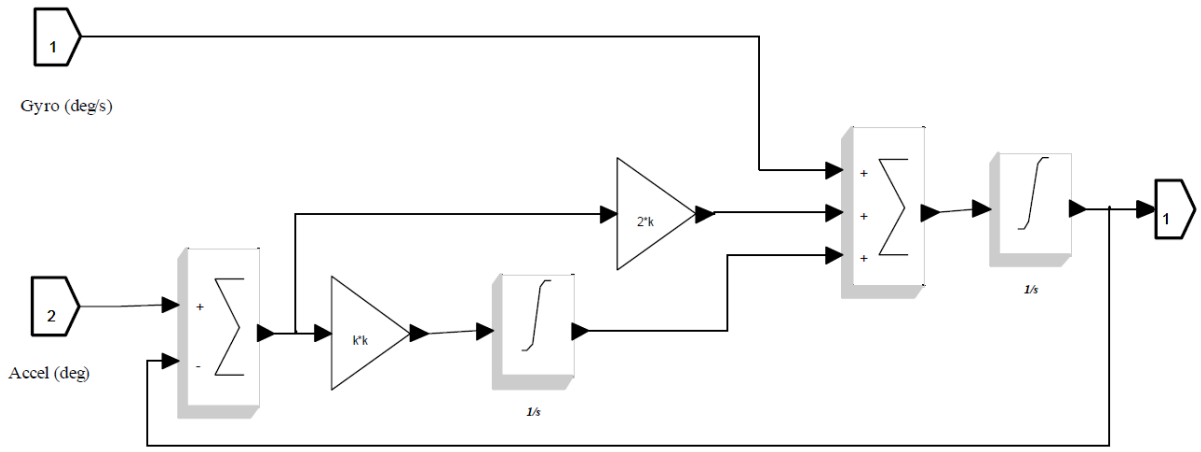

Figure 2.4: Continuous time second order filter structure

2.3.4 Low-pass Filter

It is the opposite of the high-pass filter. It passes the low-frequency signals and attenuates the ones that have a frequency higher than the cutoff frequency. It is used to isolate the force of gravity from accelerometer data. Conceptually, it is simpler than the high-pass filter [11]. Therefore, it is widely used to extract the gravity component from acceleration and subtract it to get the linear acceleration. The low-pass filter Equation 2.18 is used to get the gravitational force from the raw acceleration:

$$gravity_t = \alpha \times gravity_{t-1} + (1 - \alpha) \times acceleration \tag{2.18}$$

where alpha (α) is calculated as a/(a + dt). The constant "a" is the time-constant of the low-pass filter and dt is the delivery rate [9].

2.4 Other Techniques

There are some techniques other than sensor fusion techniques implemented in this project. This section will give an overview of these techniques and how they are used in the project.

2.4.1 Quaternion

Quaternion acts as an abstract means of representing the attitude and mainly used for angle measurement eliminating the gimbal lock problem. Gimbal lock is the inability to

get the current orientation when the pitch, roll or yaw approaches ±90 degrees. It is a four-dimensional vector used to encode any rotation in a 3D coordinate system from the *inertial frame* to the sensor *body frame* as shown below. An inertial frame is a defined fixed coordinate system of the Earth while the sensor body frame is a relative coordinate system that never changes with the sensor alignment or orientation. The Earth's coordinate axes are discussed in Subsection 2.2.1. The x-axis points to the north direction, y-axis points to the east and the z-axis points down towards the center of the Earth as shown in Figure 2.3. Technically, this vector consists of one real and three complex elements. In fact, Quaternions are more complicated and less intuitive than Euler Angles. Although there exists two attitudes for a single solution in Euler's Angle technique, Quaternion is based upon Euler's principle of rotation [10] [7].

$$q = (q_0 q_1 q_2 q_3)^T$$

where T is the Transpose operation of the vector. The elements q_0, q_1, and q_2 are the vector parts of the quaternion and the element q_3 is the scalar part that specifies the amount of rotation that should be performed on the vector part. Assume that θ is the angle of rotation and the vector $(v_x v_y v_z)^T$ is a unit vector representing the axis of rotation. The quaternion elements are defined as follows: [10]

$$\begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} = \begin{pmatrix} v_x \sin(\theta/2) \\ v_y \sin(\theta/2) \\ v_z \sin(\theta/2) \\ \cos(\theta/2) \end{pmatrix}$$

The attitude (Yaw = 0, Pitch = 0, Roll = 0) is represented in quaternion form as q = [0, 0, 0, 1]. Yaw, pitch, and roll angles can be calculated by converting Quaternions attitude estimate to Euler Angles. The conversion equations from Quaternions to Euler Angles are: [7]

$$roll = \sin^{-1}(-2(q_0 q_2 - q_1 q_3)) \tag{2.19}$$

$$pitch = \tan^{-1}\left(\frac{2(q_1 q_2 + q_0 q_3)}{q_3^2 + q_2^2 - q_1^2 - q_0^2}\right) \tag{2.20}$$

$$yaw = \tan^{-1}\left(\frac{2(q_0 q_1 + q_3 q_2)}{q_3^2 - q_2^2 - q_1^2 + q_0^2}\right) \tag{2.21}$$

2.4.2 Median Filter

The median filter is mainly used in the project as a smoothing algorithm in addition to the average method which takes the average of the current and previous values. It is a nonlinear signal processing technology based on statistics that is simply applied by replacing the noisy value of the sequence by the median of neighboring entries. Median is the middle value (in the case of odd number entries) or the average of the two middle values (in case that the number of entries is even) in a list of numbers that are sorted in ascending order. The main advantage of the median filter algorithm is noise reduction in a preprocessing step to improve data and get more accurate results. It is widely used in image processing [20].

2.5 Hardware

This section discusses the used hardware throughout the project. Smartphone Samsung Galaxy S4 is used to test the Android mobile application whereas Arduino Uno board, as well as MPU-6050 sensor are used in the embedded system application.

2.5.1 Samsung Galaxy S4

Figure 2.5: Samsung Galaxy S4 smartphone

The mobile phone used in this project is Samsung Galaxy S4 as shown in Figure 2.5. The most significant reason of choosing this device is the large number of built-in sensors into it. It contains accelerometer, gyroscope, and magnetometer sensors on which the mobile application project mainly based. Table 2.2 shows the smartphone's features and specifications [8].

Samsung Galaxy S4 Features	
Physical Dimensions	Height: 136.6 mm Width: 69.8 mm Depth: 7.9 mm Weight: 130 g
Processor	Quad-core, 1900 MHz, Krait 300
Graphics Processor	Adreno 320
Operating System	Android (5.0, 4.4.2, 4.3, 4.2.2)
System Chip	Qualcomm Snapdragon 600 8974
Memory	System Memory: 2 GB RAM Built-in Storage: 64 GB Storage Expansion: microSD, microSDHC, microSDXC up to 64 GB
Display	Physical size: 5.0 inches Resolution: 1080 x 1920 pixels Pixel density: 441 ppi
Connectivity	GSM: 850, 900, 1800, 1900 MHz Wi-Fi: 802.11 a, b, g, n, dual-band, ac microUSB 2.0 Positioning: GPS, A-GPS, Glonass Other: Near Field Communication (NFC), Tethering, Computer sync, OTA sync, ANT+, Infrared
Sensors	Accelerometer, Gyroscope, Compass, Thermometer, Gesture, Humidity, Barometer

Table 2.2: Samsung Galaxy S4 specification sheet

2.5.2 Arduino Uno

The Arduino Uno board is an open source hardware microcontroller board based on the ATmega328P powered either via the Universal Serial Bus (USB) connection or with an external power source supply. The first Arduino was introduced in 2005, aiming to provide an inexpensive and easy way for novices and professionals to create devices that interact with their environment using sensors and actuators. The ATmega328 has 32 Kilo Byte (KB), 2 KB of Static Random Access Memory (SRAM), and 1 KB of Electrically Erasable Programmable Read-Only Memory (EEPROM). Originally, the name "Uno" is an Italian word meaning one and it was chosen to mark the release of the first version of Arduino Software IDE 1.0. Besides, it is the most commonly used version of Arduino family [2]. An important aspect of the Arduino is its standard connectors, which lets users connect the CPU board to a variety of interchangeable add-on modules known as shields. Some shields communicate with the Arduino board directly over various pins, but many shields are individually addressable via a serial bus. Many shields can be stacked and used in parallel. It is used in this project to connect the sensor MPU-6050

discussed in Section 2.2 with the Personal Computer (PC) and can be programmed with the Arduino Software IDE which will be described in the next section. Features and technical specifications of Arduino Uno board are shown in Table 2.3.

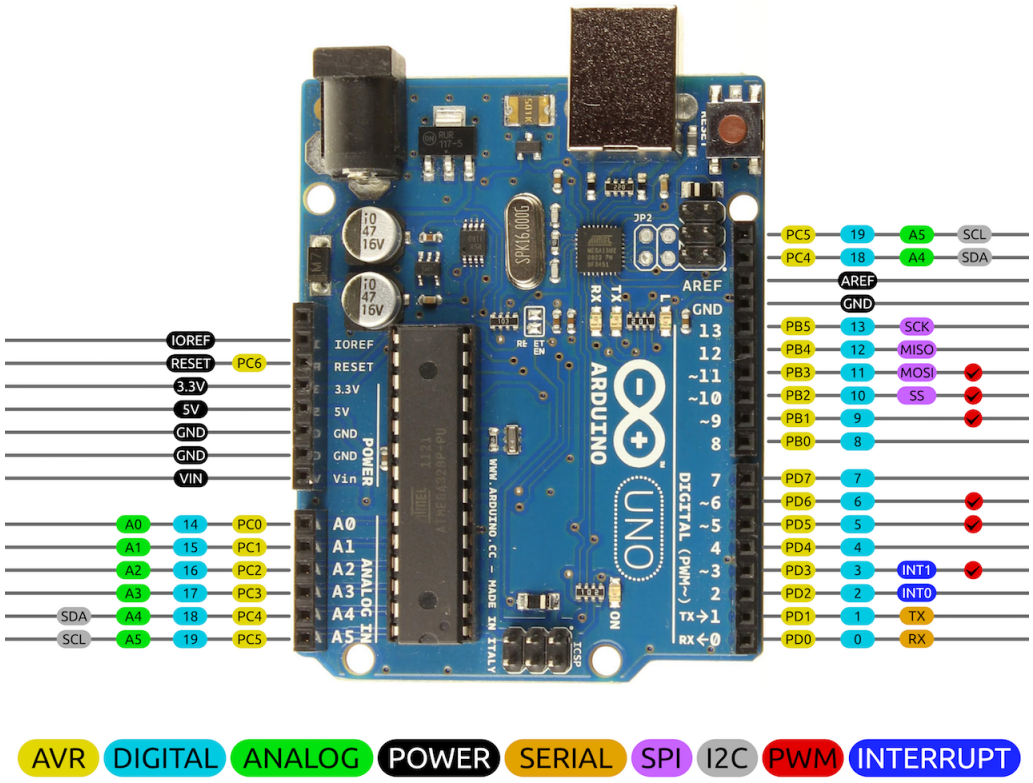

Figure 2.6: Arduino Uno board

2.6 Platform

This section gives a brief overview of the platforms and programs used in this project. Android operating system is used in developing mobile applications that the users can run on their smartphones. It is used in the project to read data from sensors in android devices. Moreover, the purpose of Arduino and processing IDE is to communicate with Arduino Uno board and sensor MPU-6050 which are used in the embedded system project.

2.6.1 Android

Android is a mobile operating system founded in Palo Alto, California by Andy Rubin, Rich Miner, Nick Sears, and Chris White in October 2003. It was later bought by Google in 2005. It was designed for touchscreen mobile devices such as smartphone and tablets and Google further developed further developed Android TV for televisions, Android

Arduino Uno Board	
Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage	7-12V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Table 2.3: Arduino Uno technical specifications

Auto for cars, and Android Wear for wrist watches. In addition, it is based on the Linux kernel [13]. In July 2013, the Google Play store had over one million Android applications published and in September 2015, Android had 1.4 billion monthly active devices. It is used in this project to read data from smartphone sensors using Android Studio as a software and Java as a programming language. Then, the application displays the measured and computed data with the option to record and save them in Comma Separated Value (CSV) files. Moreover, sensor fusion techniques are applied on the sensors raw data and a demo in the application shows the output after improvements.

2.6.2 Arduino IDE

The founders of Arduino project used to meet in a bar in Ivrea, Italy called Arduino, so the idea of the name Arduino came from here. It is released in 2005 and became a widely used open-source electronics platform nowadays [12]. It is used in Internet of Things (IoT) applications and embedded systems. It is simple and based on a simple micro-controller board and a development environment IDE for writing software for the board such that the Arduino board can be connected to the computer and read data from the sensors that are connected to it such as the MPU-6050 sensor used in this project. Arduino board designs use a variety of microprocessors and controllers and they are equipped with sets of analog and digital Input/Output (I/O) pins. The received data is then processed using Arduino Programming Language and can be sent to another

program using serial communication such as I2C protocol. It is a cross-platform IDE that runs on Mac, Windows, and Linux [1].

2.6.3 Processing IDE

Processing is an IDE and open source computer programming language that was initiated in 2001 by Casey Reas and Benjamin Fry. The Processing Development Environment (PDE) includes a text editor to write processing program in a sketch. Then, the program is compiled using the compiler and the output is displayed in a display window or the text console when the run button is pressed. It has different programming modes so that sketches can be deployed on different platforms [6]. It is used in the project to receive data serially from Arduino IDE and export them in a CSV file in order to save and plot the recorded data using Microsoft Excel and evaluate sensors performance. Last but not least, an interactive visualization example of the data showing the effect of motion of MPU-6050 sensor with and without sensor fusion is developed as an output.

2.7 Related Work

2.7.1 Indoor Positioning using Sensor-fusion in Android Devices

The main aim of this thesis is to increase reliability and precision of indoor localization using built-in sensors in cellular phones. Movement, pace, and heading indoors can be detected using multisensor fusion. For example, the distance is calculated using the accelerometer by double integrating the measured linear acceleration and the gyroscope keeps track of the orientation of the device. These calculations are enhanced using walking speed limit and step detection to reduce the error. Then, these data are integrated with signal strength measurements of the Wi-Fi such that the current position can be estimated by comparing the measured signal to the previously stored fingerprints in the database and selecting the nearest values. The final results showed that the average deviation between the estimated and real position was less than two meters [18]. Figure 2.7 shows the data flow digram for the application. It shows the states and the sequences between them in a typical scenario.

2.7.2 An Introduction to Multisensor Data Fusion

The objective of this paper is to provide an introduction to the fusion of multisensor data as a basis for further study and research. In fact, sensor fusion technology is not only

[1] https://www.arduino.cc/en/Guide/Introduction

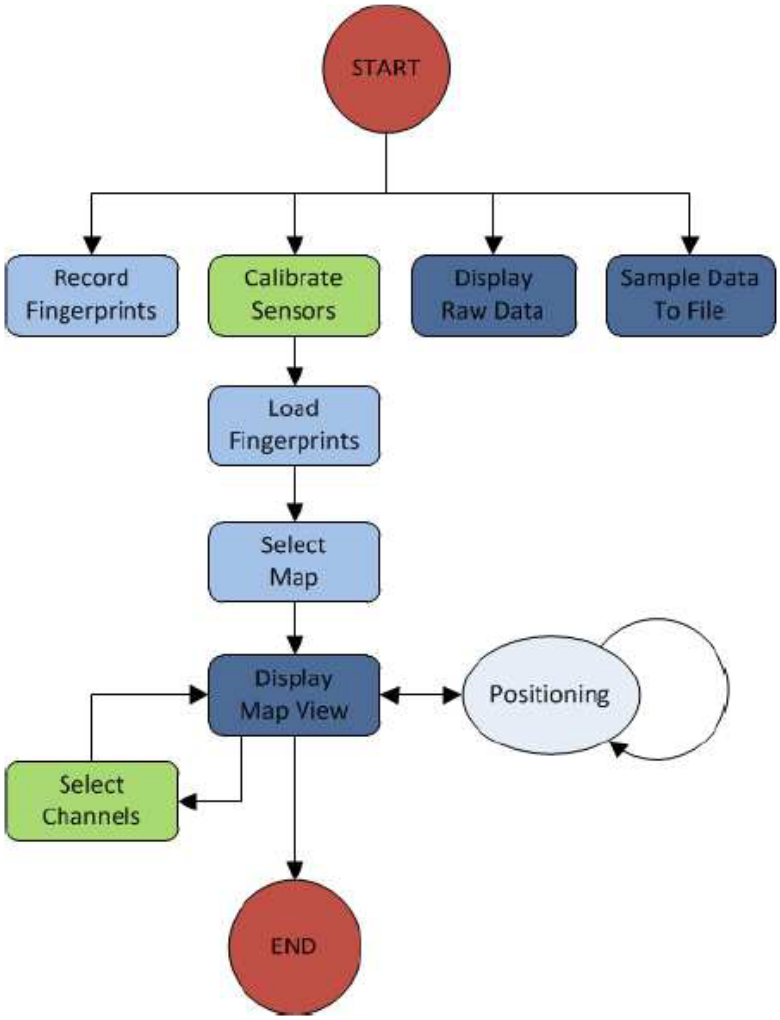

Figure 2.7: Application state chart

applied to Department of Defense (DoD) areas like automated target recognition, but also to nonmilitary applications such as medical diagnosis. In addition, an overview of several data fusion techniques along with a discussion of some fundamental issues as well as assessment of the state-of-the-art and state-of-practice is provided [16].

2.7.3 An Introduction to Sensor Fusion

This research is intended to introduce sensor fusion principles and methods like Kalman filter and interference methods. A survey of the architecture in addition limitations of sensor fusion are presented. Moreover, the difference between sensor fusion and integration as well as the most common sensor fusion applications are discussed [19].

2.7.4 Multisensor Fusion and Integration: Approaches, Applications, and Future Research Directions

The synergistic combination of sensory data from several sensors to obtain more reliable and accurate information is referred to as multisensor integration and fusion. The aim of this paper is to introduce the paradigm of sensor fusion and integration. Furthermore, emerging applications in areas like robotics, remote sensing, biomedical systems, and transportation are surveyed. Finally, a discussion about promising future research areas including multilevel sensor fusion, sensors fault detection, and adaptive multisensor fusion is provided [5].

2.7.5 Pedestrian Indoor Positioning and Tracking using Smartphone Sensors, Step Detection, and Map Matching Algorithm

In this thesis, inertial sensors such as acceleration sensors, gyroscopes, and magnetometers, that are built in a cellular phone, are used for indoor navigation. The orientation of the pedestrian, as well as the distance, need to be calculated in order to get the pedestrian position. Despite the fact that the accuracy of these sensors deteriorates due to the accumulation of inertial measurements errors in the integration process, several algorithms to determine pedestrian trajectory movement are proposed and the accuracy of the output result is significantly increased. For example, the Map Matching method was implemented into the calculation process to increase the long-term accuracy in the determination of the pedestrian position [21]. Figure 2.8 shows the scheme of the whole process.

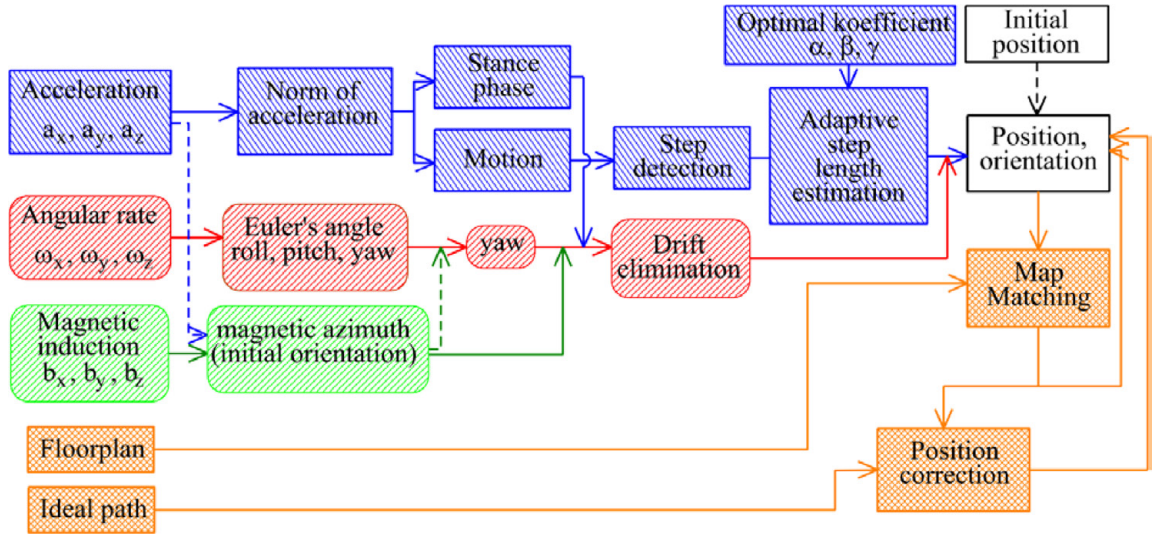

Figure 2.8: Flowchart of the processing

Chapter 3

Methodology

This chapter discusses the software prototype implementation. Besides, It contains a detailed description of the development and functionality of the Android application and the Arduino project. To begin with, the different methods used for improving the performance of the sensors are discussed in the first section. Then, the flow and calculation of data from sensors readings are explained. Finally, screenshots with a detailed demonstration for both applications are attached.

3.1 Operation Modes

There are eight modes supported by both applications and each mode represents certain feature or technique.

1. **Raw Data Mode:** In this mode, only raw measurements that are received from the sensors are displayed and used to calculate the angle in case that the gyroscope is used or yaw, roll, pitch, velocity and displacement in case of using the accelerometer.

2. **Average Subtraction Mode:** Average linear acceleration and angular speed calculated from previous tests are subtracted from accelerometer and gyroscope measurements.

3. **Noise Reduction Mode:** The aim of this mode is to get rid of the noise caused by external factors and the surrounding environment when the device is motionless. It is used with raw data or average subtraction mode. There is no update in the rotation angle or displacement unless the change is greater than a specific constant value epsilon. Epsilon is a filter-threshold for discarding gyroscope or accelerometer measurements that are below a certain level and potentially are only noise and not real motion. The value of this constant is calculated as follows:

$$\epsilon = \sqrt{x^2 + y^2 + z^2}$$

where x, y, and z are the angular velocity or acceleration readings for the three axes.

4. **Average Smoothing Mode:** This mode applies a smoothing technique to raw data or average subtraction mode and can be used with median filter as well. The average smoothing technique is simply taking the average of the current and the previous acceleration or angular velocity. Then, these calculations are used to process all the other data. This method improves the performance by reducing the sharp fluctuations and makes the plotted graph smoother.

5. **Median Filter Mode:** The median filter is a smoothing algorithm discussed in Section 2.4.2 is applied in this mode. This technique can be used with any other one or more modes.

6. **First Order Complementary Filter Mode:** First order Complementary filter is a sensor fusion technique used to combine data from several sensors to get more accurate results. It is discussed in details in Section 2.3.2 and is implemented in this mode. This mode in addition to Kalman filter mode are the optimum features in this project as it is combined with average subtraction mode and it can include any of the smoothing techniques to obtain the most accurate information.

7. **Second Order Complementary Filter Mode:** This mode has the same features as the previous one except that it is implemented with the second order Complementary filter. You can refer to Section 2.3.2 for more details.

8. **Kalman Filter Mode:** Kalman filter is an improvement algorithm that fuses data from multi sensors to get more accurate information. It is discussed in details in Section 2.3.1 and implemented in this mode. It is combined with average subtraction mode and can include any of the smoothing techniques to obtain optimum output.

3.2 Data Calculation

3.2.1 Gyroscope Data Calculation

The gyroscope in Android Section 2.1.2 and the MPU6050 gyro in Section 2.2.2 are used to get the angle of rotation in the 3 dimensions by integrating the angular velocity measured from them. The rotation angle is calculated differently for each gyroscope. In Arduino project, the angle is calculated as follows:

$$angle_t = \int angular\_velocity.dt = angle_{t-1} + angular\_aelocity \times \Delta t$$

where Δt is the delivery rate. In Android application, some predefined methods that require extra steps are used.

Listing 3.1: Gyroscope angles calculation in Android

```
// convert the raw gyro data into a rotation vector
getRotationVectorFromGyro(gyro, deltaVector, dt / 2.0f);

// convert rotation vector into rotation matrix
SensorManager.getRotationMatrixFromVector(deltaMatrix,
        deltaVector);

/* apply the new rotation interval on the gyroscope based
rotation matrix */

gyroMatrix = matrixMultiplication(gyroMatrix, deltaMatrix);
SensorManager.getOrientation(gyroMatrix, gyroOrientation);
```

The code in Listing 3.1 is used to get the angle from the gyroscope data in Android. The method *getRotationVectorFromGyro* takes data from the gyroscope, delivery rate and the rotation vector that will contain the result as parameters. This method integrates around each axis with the angular speed by the time-step. In order to get a delta rotation from this sample over the time-step, this axis-angle representation of the delta rotation will be converted into a quaternion before turning it into the rotation matrix. The method *matrixMultiplication* is implemented to multiply two given matrices and it is used in the code to multiply the new result matrix by the old one and save the result to be used later again. Finally, the predefined methods *getRotationMatrixFromVector* and *getOrientation* are used to get rotation matrix from rotation vector and apply the new rotation interval on the gyroscope based rotation matrix respectively. The final result is the array *gyroOrientation* and it contains the rotation angles in the X, Y and, Z directions in indexes 0, 1 and 2 respectively.

3.2.2 Accelerometer Data Calculation

The accelerometer in Android Section 2.1.1 and the MPU-6050 accelerometer in Section 2.2.1 are used to get the velocity and displacement in the 3 dimensions by integrating the acceleration values measured from them. The raw acceleration and the way of velocity and displacement calculation are the same for both applications. However, the method to get the linear acceleration is different. The sensor MPU-6050 measures raw acceleration containing gravity component. In order to get the linear acceleration, low pass filter algorithm has to be implemented and applied to the raw accelerations in the X, Y, and Z axes to extract the gravity component. Then, the calculated gravity value is subtracted from the raw acceleration to obtain the linear acceleration in all directions. Regarding Android accelerometer, linear acceleration can be obtained directly by using *TYPE_LINEAR_ACCELERATION* sensor. The velocity is calculated from linear

acceleration by integration.

$$velocity_t = \int linear\_acceleration.dt = velocity_{t-1} + linear\_acceleration \times \Delta t$$

while the displacement is obtained from linear acceleration by double integration i.e. integrating the calculated velocity at this time.

$$displacement_t = \int\int linear\_acceleration.dt = displacement_{t-1} + velocity \times \Delta t$$

In addition, orientation angles can be calculated from accelerometer data. Yaw, pitch and roll can be calculated using MPU-6050 sensor directly using the equations mentioned in Section 2.2.1 or can be calculated using quaternion which is explained in Section 2.4.1. Regarding the mobile application, azimuth, pitch and roll are calculated using *TYPE_ACCELEROMETER* and *TYPE_MAGNETIC_FIELD* sensors as shown in the following code in Listing 3.2.

Listing 3.2: Orientation angles calculation in Android

```
/* Check if this condition returns true and there is a not null
result in matrix variable. accel and mag arrays contain
accelerometer and magnetometer data respectively. */

if (SensorManager.getRotationMatrix(matrix, null, accel, mag))
 SensorManager.getOrientation(matrix, accMagOrientation);

// azimuth = accMagOrientation[0]
// pitch = accMagOrientation[1]
// roll = accMagOrientation[2]
```

3.3 Android Application

This section describes the mobile android application flow. Firstly, there is a welcome home view containing a navigation drawer which displays all the application implemented techniques as in Figure 3.1. In the navigation drawer, there are translation and rotation modes and each mode has a list containing several algorithms. There are two ways to display data for each technique such that the user can either view the sensors measured and calculated values or represent these values as a demo. Moreover, there are two options in this application. There is a button to reset the timer and all data values immediately and another button to record data for 3 minutes. The record button resets all data and starts recording 30 seconds after the button click. This is because the fluctuations caused by the button, when it is being pressed, has to be removed. Later after the 3 minutes recording, the data are saved automatically to a CSV file for further data processing. Smoothing algorithms are represented as checkboxes in all rotation and translation modes. When the checkbox is selected, the technique is activated immediately. Many checkboxes can be selected.

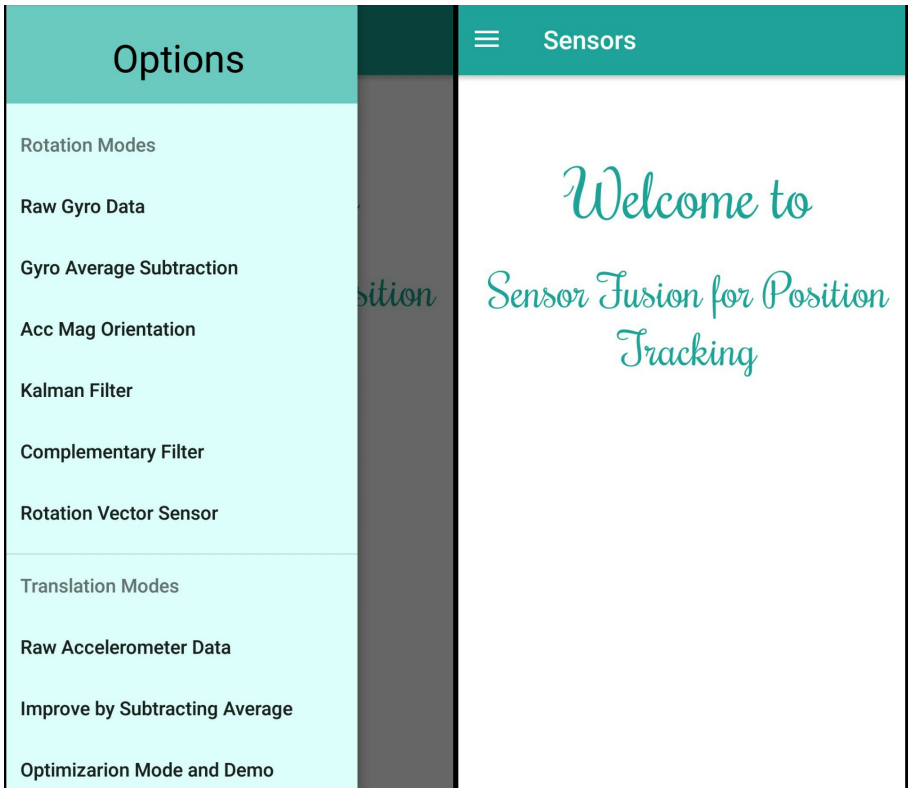

Figure 3.1: Welcome view and navigation drawer

3.3.1 Rotation Modes

There are seven rotation modes in this application: raw gyroscope data, average subtracted gyroscope, accelerometer magnetometer orientation, accelerometer gyroscope orientation, Complementary filter, Kalman filter, and rotation vector sensor. You can refer to Section 3.1 for more details about each technique. Furthermore, the user has the option to apply either median filter algorithm or average smoothing technique individually or together for each mode to eliminate noise. To begin with, average subtraction is an improvement for the raw data mode. All the other rotation modes that are using gyroscope are dependent on average subtraction mode. Accelerometer magnetometer orientation uses *TYPE_ACCELEROMETER* and *TYPE_MAGNETIC_FIELD* sensor in order to get the orientation angles in the X, Y, and Z directions. This can be done as shown previously in Listing 3.2. Moreover, accelerometer gyroscope orientation uses *TYPE_ACCELEROMETER* as well as *TYPE_GYROSCOPE* sensors and applies Complementary filter on them as done in the Arduino project to obtain 3D orientation. Complementary and Kalman filters are using all the hardware sensors in Android which are the accelerometer, gyroscope and magnetometer while the rotation vector mode uses the virtual *TYPE_ROTATION_VECTOR* sensor which is explained in Section 2.1.4. Figure 3.2 is a data flow diagram that shows all the modes sequence in the mobile application. For each mode, there are two options. You can either view the current time and data values (this is the default mode) or switch to demo mode by clicking on the "DEMO"

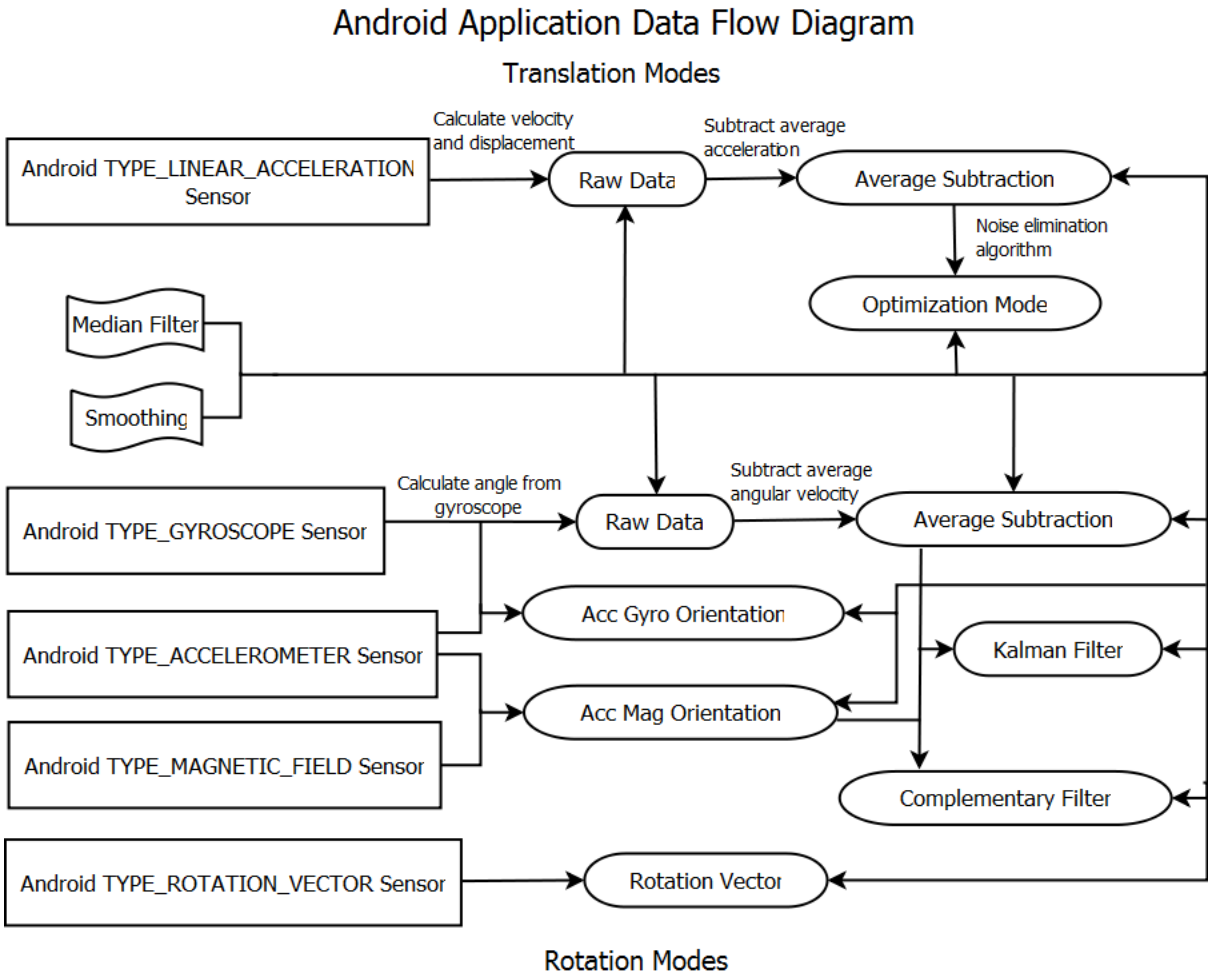

Figure 3.2: Android application data flow diagram

button and see the effect of change of rotation angles on the textured cube and how the improvement and smoothing techniques has a powerful impact. You can switch back from the demo mode to the original mode by pressing back in your smartphone. A screenshot visualizing both modes is shown below in Figure 3.3.

3.3.2 Translation Modes

There are three translation modes in this application: raw data, average subtraction, and optimization and demo mode. Average smoothing and median filter can be used with each of these modes by selecting the checkboxes. The first two modes are the same techniques applied in rotation, but using the accelerometer sensor (*TYPE_LINEAR_ACCELERATION*) rather than the gyroscope sensor (*TYPE_GYROSCOPE*). The last mode is an improvement to the average subtraction technique. The main purpose of the algorithm that is implemented in this mode is to enhance the displacement calculation from the accelerometer and eliminate noise. The method is that the distance and velocity are not calculated

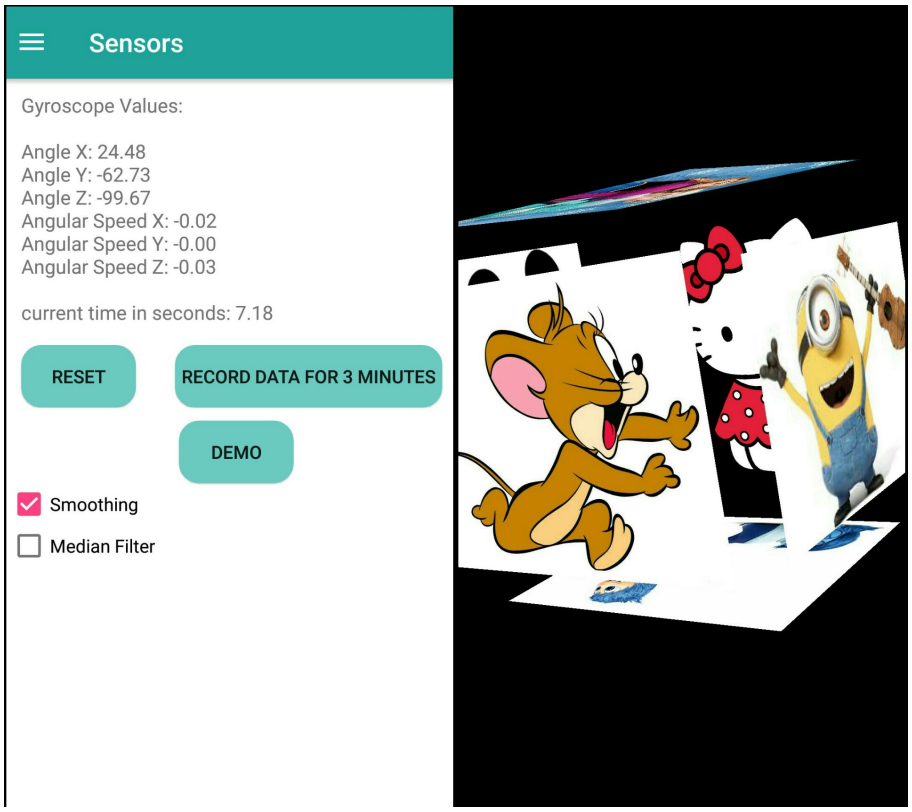

Figure 3.3: Rotation data view and demo

unless there is a real motion detected. This is done by calculating a constant from the accelerometer data and check if this constant is greater than certain value (EPSILON) as shown in the code snippet in Listing 3.3 below. The EPSILON value is determined by taking the average of several experiments. If this condition is satisfied, the velocity and displacement are calculated normally. Otherwise, the velocity is set to zero so that the displacement will not change.

Listing 3.3: Accelerometer optimization algorithm in Android

```
omega = accelerationX^2 + accelerationY^2 + accelerationZ^2

// dt_acc is delta time for the accelerometer sensor

IF (omega > EPSILON) THEN
   velocityX += accelerationX * dt_acc
   velocityY += accelerationY * dt_acc
   velocityZ += accelerationZ * dt_acc
ELSE
   velocityX = 0
   velocityY = 0
   velocityZ = 0
```

```
ENDIF

displacementX += velocityX * dt_acc
displacementY += velocityY * dt_acc
displacementZ += velocityZ * dt_acc
```

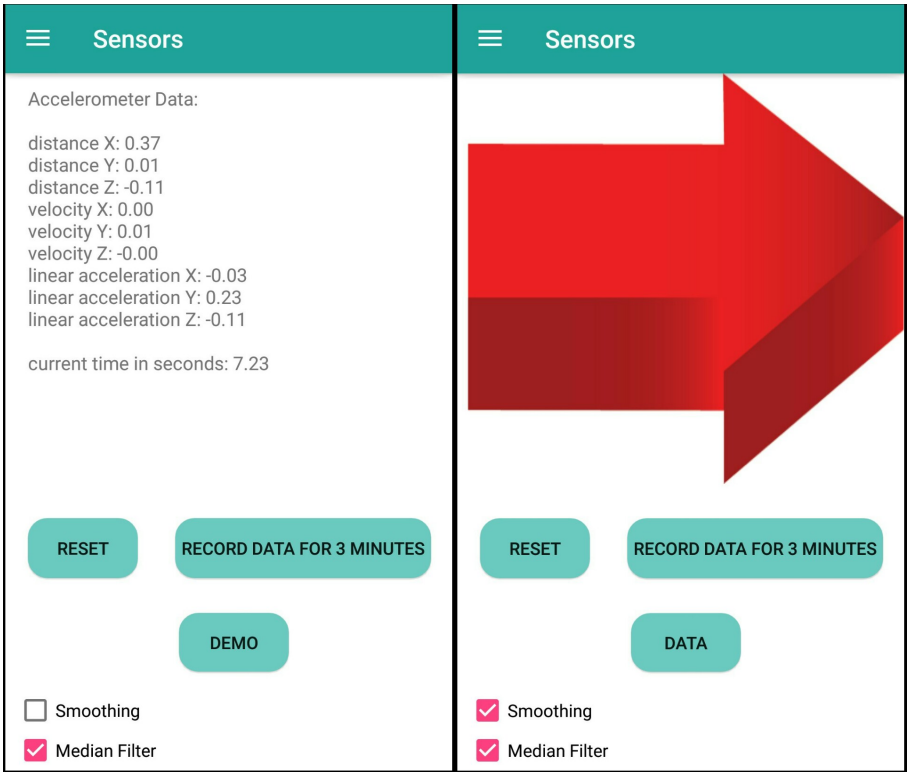

Figure 3.4: Translation data view and demo

The first two modes have only data view option while the last one has a demo feature as well. Figure 3.4 demonstrates the data and demo views in the Android application. In the demo mode, the current direction is detected according to the accelerometer data that is determined by the device movement. This is done by calculating the difference between the current and the previous linear acceleration. Then, the axis is determined according to the largest absolute difference and the sign of this acceleration difference (positive or negative) indicates the the movement direction of that axis (right or left for example in case of the x-axis). The current direction will not be changed due to any unexpected fluctuations. This is handled by using a very small delay in case of sudden direction change. During this delay, we will keep track of the direction change only without modifying the current direction to make sure that the direction is really changed and it is not just noise. The direction can be changed if there is still change in direction after the delay or the state of the accelerometer is changed from its initial state at rest to motion. Listing 3.4 contains part of the algorithm in pseudocode showing how the

technique is implemented. This code only explains the procedure of detecting motion in the x-axis direction. However, the actual full code in the project applies the same algorithm in the Y and Z axes as well.

Listing 3.4: Direction detection method

```
// acceleration difference calculation
Ax  = accelerationX - last_accX
Ay  = accelerationY - last_accY
Az  = accelerationz - last_accZ

/* Check if there is change in direction after the delay or
 motion after rest. Delay and motion variables represent number
 of readings, so 3 readings (approximately 0.3 seconds) are
 skipped as a delay and 1 reading (0.1 s) to make sure that
 there is a real motion detected. The motion variable is
 incremented when direction = 0 i.e. no motion */

IF(|Ax| >= |Ay| AND |Ax| >= |Az|) THEN
  IF(Ax >= 0) THEN // positive change
    d = 1 // the supposed direction
    IF((direction != d AND delay >= 3) OR
        (direction = 0 AND motion >= 1)) THEN
          direction = 1
          delay = 0
          motion = 0
    ELSE
         IF(direction != d AND delay < 3) THEN
           delay = delay + 1
         ENDIF
    ENDIF
  ELSE // negative change
    d = -1
    IF((direction != d AND delay >= 3) OR
        (direction = 0 AND motion >= 1)) THEN
          direction = -1
          delay = 0
          motion = 0
    ELSE
         IF(direction != d AND delay < 3) THEN
           delay = delay + 1
         ENDIF
    ENDIF
  ENDIF
ENDIF
```

3.4 Arduino and Processing Project

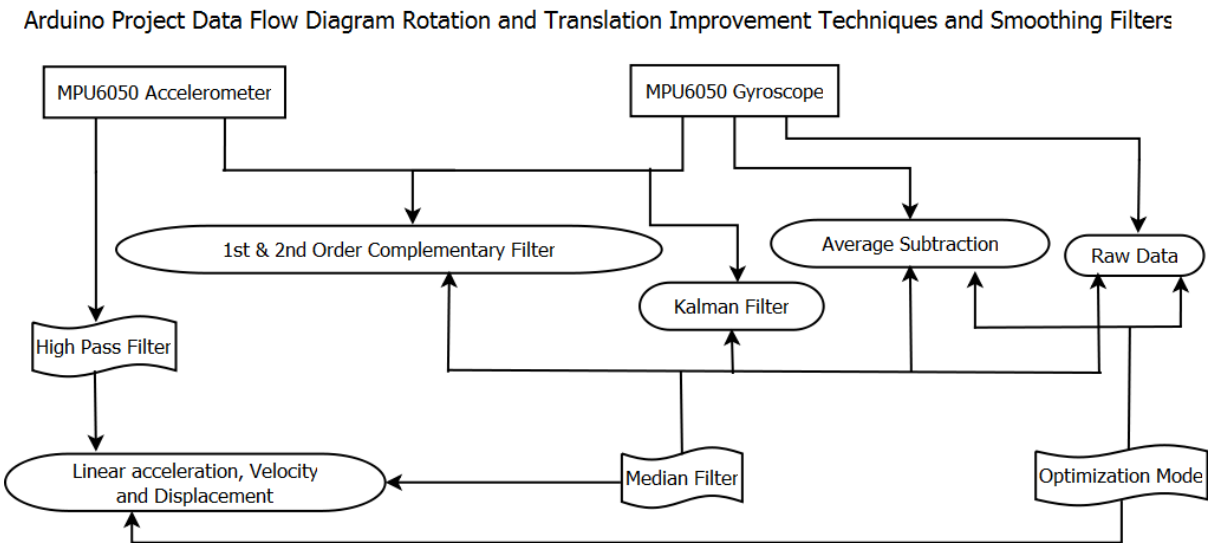

Figure 3.5: Arduino project data flow diagram

The main purpose of this section is to discuss the flow of Arduino and Processing project. The first subsection describes the Arduino MPU-6050 sensor wiring up. Then, the project and its procedures are explained. Figure 3.5 is a data flow diagram that represents how the improvement techniques and filters applied in the Arduino and Processing project.

3.4.1 Connection

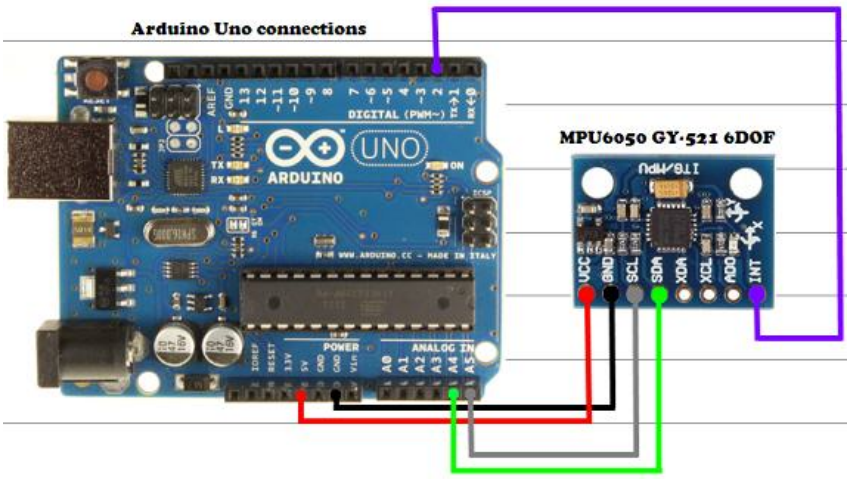

Figure 3.6: MPU-6050 connection with Arduino Uno

Arduino and Processing IDE in Section 2.6 mainly are the software used to implement this project. The hardware used are Arduino Uno board in Section 2.5.2 and the sensor MPU-6050 which is described in Section 2.2. They are connected together as shown in Figure 3.6. The MPU-6050 module has a 5V pin that is connected to the Arduino's 5V or 3.3V pin. Then, the GND of the Arduino is connected to the GND of the MPU-6050 and Arduino's digital pin 2 (interrupt pin 0) is connected to the pin labeled as INT on the MPU-6050. Last but not least, connect the pin labeled SDA on the MPU-6050 to the Arduino's analog pin 4 (SDA) and the pin labeled as SCL on the MPU 6050 to the Arduino's analog pin 5 (SCL) in order to set up the I2C lines.

3.4.2 Project Implementation

Figure 3.7: Processing welcome view

After the sensor is connected to the Arduino board, an Arduino code that reads data from the gyroscope and accelerometer is uploaded to the board. This code only reads data from MPU-6050 versus time and sends it serially to Processing program for further data calculation and visualization using the demo. After compiling Processing code successfully, a welcome view containing two options appears and the program begins to receive serial data from Arduino. The user can select any of the two modes either rotation or translation in order to start viewing data and demo or press 'ESC' in the keyboard to exit as shown in Figure 3.7. After choosing an option, the user will be redirected to a new view where the calculated data (angle in the case of rotation and displacement when choosing translation mode) in the three directions are displayed and the axes positive

directions are shown. Moreover, there are several options such as returning back to the welcome view by pressing key 'B'. All data and timer can be reset using the 'R' key. More importantly, all the data including the values received from Arduino, the calculated parameters and the current time are recorded starting from mode activation at the beginning or changing the type of algorithm in the same mode till the user press 'E' key to end the program. And hence, the measurements are saved to an Excel sheet to be used later for statistics or comparison between various techniques.

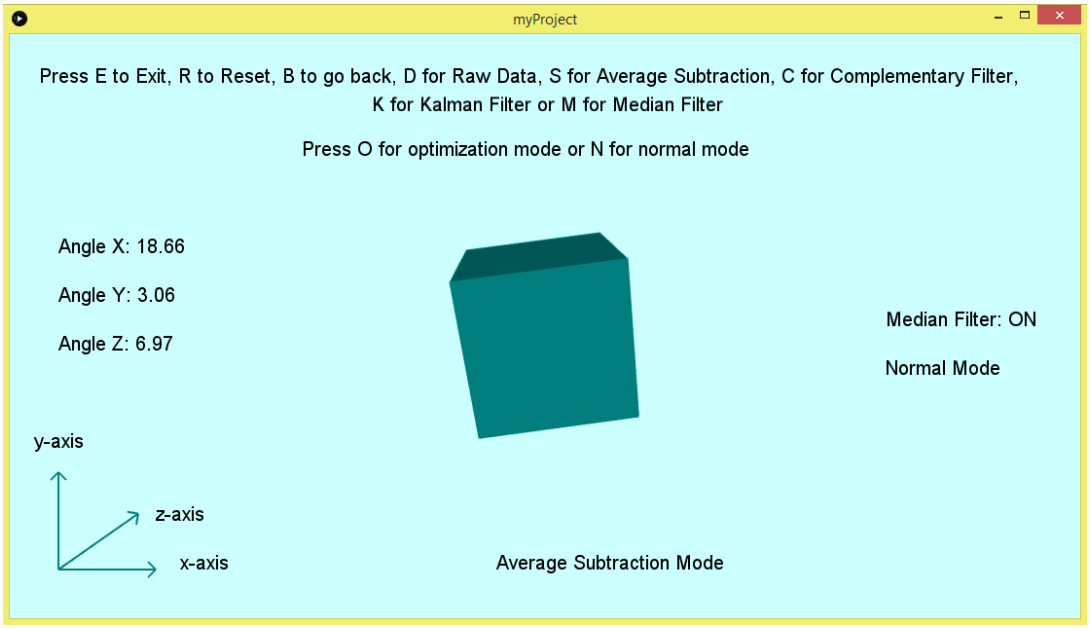

Figure 3.8: Processing rotation mode view

In the rotation mode, calculated angles in all directions are displayed and a cube is rotated in the X, Y, and Z directions by the corresponding angle value in each axis. This project includes four improvement algorithms, in addition to the raw data mode, which are first and second order Complementary filter, average subtraction, and Kalman filter. Furthermore, the two implemented smoothing techniques, median filter and smoothing by the average method, can be applied to any of these modes individually or together. Optimization mode is an extra feature used with the raw data and average subtraction modes only. This mode is simply eliminating noise. This is done by checking that the angular velocity exceeds a certain threshold value. In this case, the angles are calculated. Otherwise, the sensors measurements are ignored and hence, there will be no change in calculated data and the error percentage is reduced. All these modes are previously discussed in details in Section 3.1. Figure 3.8 show a screenshot for the application's rotation mode.

On the other hand, the translation view has only one mode which depends on applying low pass filter on the raw accelerometer data to extract the gravity component and subtract it

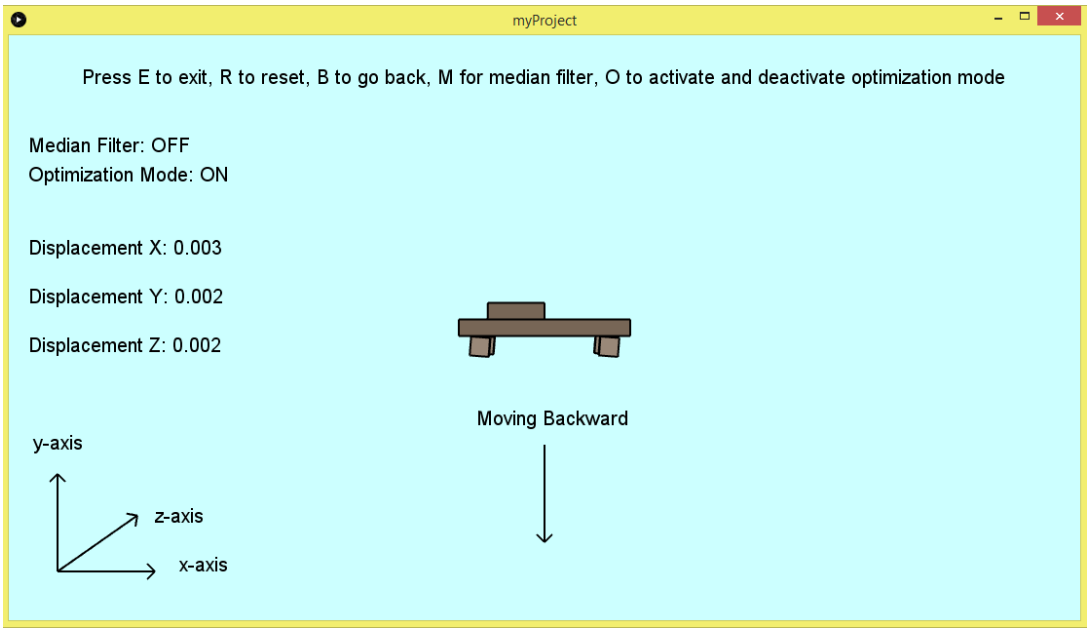

Figure 3.9: Processing translation mode view

from the acceleration. In the next step, velocity and displacement are calculated from the linear acceleration as explained before in Section 3.2.2. Median filter and noise reduction algorithms are implemented and can be used individually or together to get more accurate displacement value and reduce the error percentage. The demo for this mode is a simple car and according to the sensor motion, the direction could be determined using the same method discussed in the Android mobile application in Section 3.3.2 and an arrow representing the movement direction is displayed as shown in Figure 3.9. Last but not least, Table 3.1 below contains the keys used in the application for techniques activation in rotation and translation modes.

Key	Mode Activated
C	Complementary Filter (Rotation Only)
D	Raw Data
S	Average Subtraction
M	Median Filter
K	Kalman Filter (Rotation Only)
O	Optimization Mode (And Normal Mode as well for Translation)
N	Normal Mode (Rotation Only)
B	Back to Welcome View
R	Reset
E	Save Data and Exit

Table 3.1: Processing application manual

Chapter 4

Experimental Results

This chapter includes the results of the experiments that are done in this project using MPU-6050 sensor and smartphone sensors. The following experiments show how the improvement and smoothing techniques, that are implemented in the project, reduces the drift and error percentage significantly. Stationary and motion tests are the two testing ways that are used to investigate the accuracy of multisensor fusion algorithms as well as the performance of noise filtering techniques. The cellular phone is fixed at rest in the stationary test. On the other hand, it is rotated around a certain axis and returned back to its initial position in the motion test. The results are demonstrated in two different ways. The first one is by plotting graphs from recorded data and the other way is taking data samples from the same recorded data at seconds 1 and 61. This is to eliminate any fluctuations or noise that occur at the beginning of the experiment. Then, the error during this one-minute interval is calculated. Eventually, the average error for each method is taken and hence, different enhancement algorithms can be compared and determine whether the results are truly improved or not.

4.1 Stationary Device Test

In the first test method, the smartphone or the hardware sensors are motionless and lay flat with their back on the table. Samples are taken for 3 minutes and the recorded data are saved to a CSV file at the end of the interval in order to plot graphs and calculate the average error. The phone and the sensor are immobile to prevent any force other than gravity from affecting the output.

4.1.1 MPU-6050 Sensor Result

Firstly, the sensor is fixed in a breadboard in order to guarantee that there is no motion while testing. Then, the data is recorded for 3 minutes and the graphs are plotted. Figure 4.1 is comparing the raw angular velocity received from the gyroscope versus

after applying the average subtraction technique. It is obvious that there is a remarkable improvement as the graph approaches zero as shown below in graph B. Graph A represents the measured sensor data.

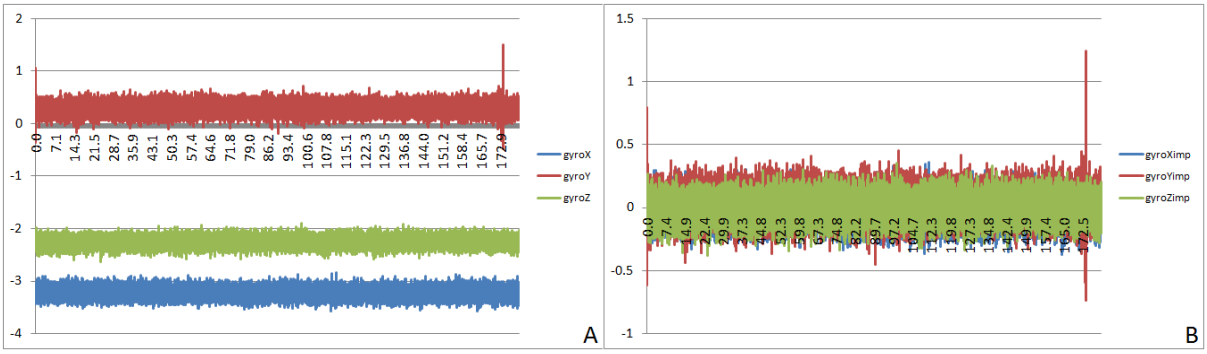

Figure 4.1: Raw angular velocity is plotted in graph A while graph B shows the average subtracted angular velocity

Graph A in Figure 4.2 shows the angles calculated in the X, Y, and Z directions from the raw angular velocity while graph B shows the calculated angles from average subtracted angular velocity. There is a noticeable error reduction where the error in the angle in the X direction is 580.6° while the average subtracted angle error in the same direction is 4.6°.

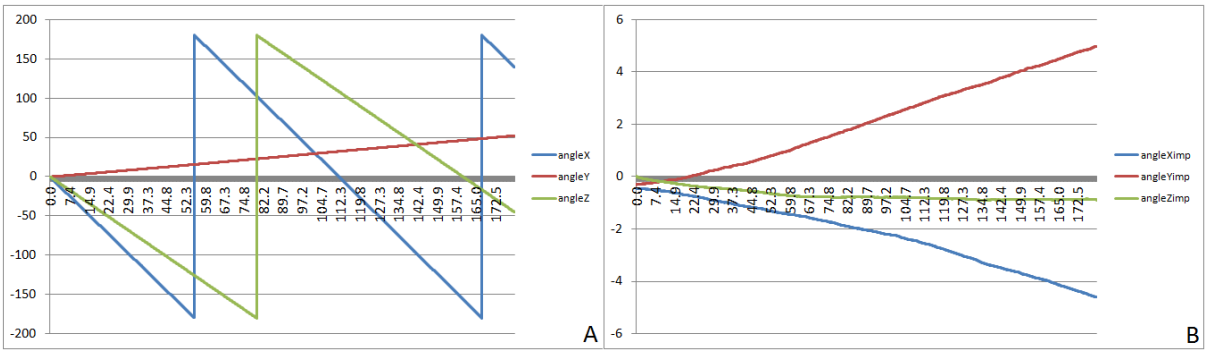

Figure 4.2: Graph A is the calculated angles from Raw data and graph B represents the computed angles from average subtracted data

Figure 4.3 shown below contains two acceleration graphs. The first one (A) is the raw acceleration measured from the accelerometer. This acceleration includes the gravitational force component. Since the sensor is fixed such that its back faces the table, therefore the gravity acts on the sensor's z-axis. And hence, the acceleration in the z-axis will be approximately equal to 0 - (-9.81) = 9.81 m/s^2 while the X and Y acceleration components are around 0 m/s^2 as there is no motion or force of gravity acting on them. The second graph, labeled as B, shows the result linear acceleration after applying high pass

filter on the raw acceleration. Apparently, the gravity component is removed resulting in approximately zero linear acceleration graph in all directions and that makes sense because there is no motion and the real acceleration is 0 m/s^2.

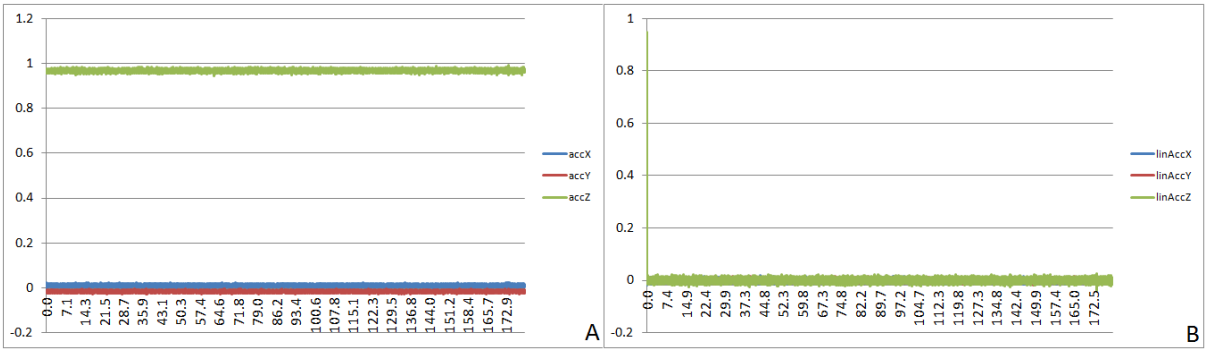

Figure 4.3: Raw and linear acceleration graphs are labeled as A and B respectively

A graph of the velocity, which is calculated from the raw acceleration by integration, is plotted in Figure 4.4. Graph A represents the velocity graph in the x-axis and y-axis while the other one, labeled as B, shows the velocity graph in the z-axis. The graphs are split because the scale of the velocity graph in the Z direction is much greater than that in the X and Y directions. The same applies to the displacement graphs in Figure 4.5. In fact, the plotted displacement data is calculated from the raw acceleration by double integration.

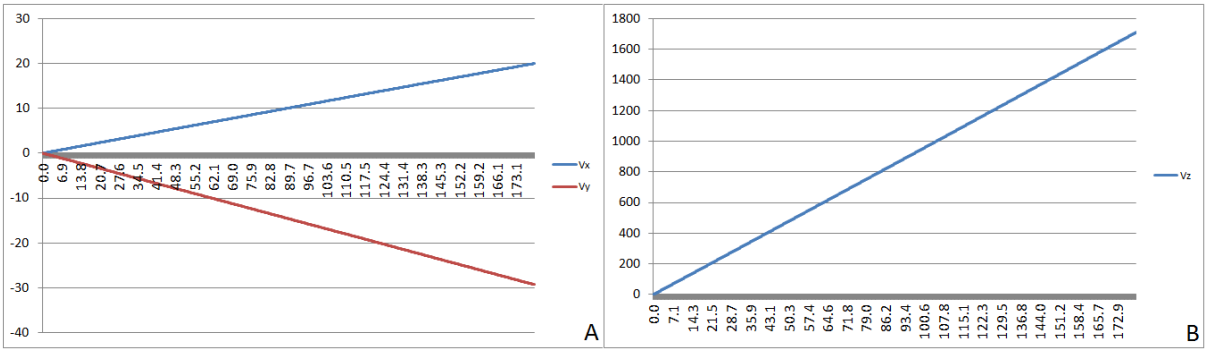

Figure 4.4: Raw velocity graphs in the X and Y directions are shown in graph A while the Z direction is plotted in graph B

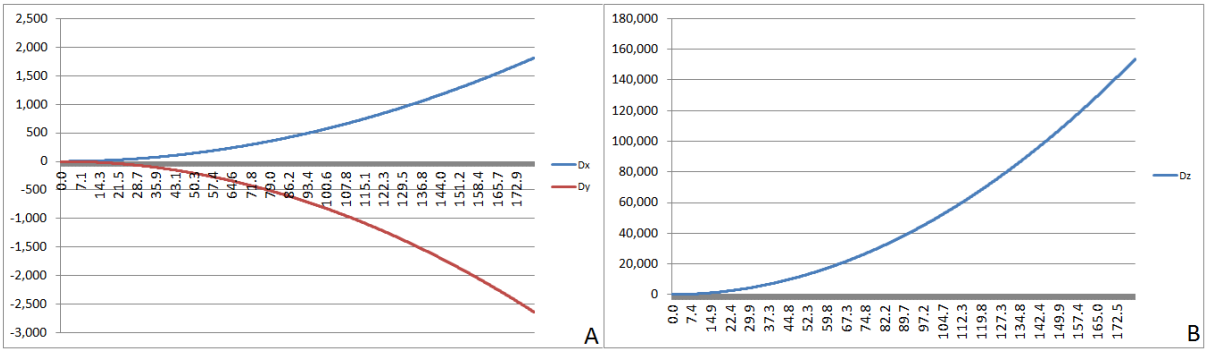

Figure 4.5: Raw displacement graphs in the X and Y directions are shown in graph A while the Z direction is plotted in graph B

Graph A in Figure 4.7 shows the velocity, that is calculated from the linear acceleration after applying the high pass filter on the raw acceleration data, versus time. In addition, the plotted graph in Figure 4.6 represents the displacement, that is calculated from the linear acceleration, versus time. There is a noticeable variation between the raw and the improved displacement. For example, the error in the raw displacement in the x-axis is 1813.5 meters whereas the error in the improved displacement in the same direction is 0.027 meters. Therefore, the error is remarkably diminished.

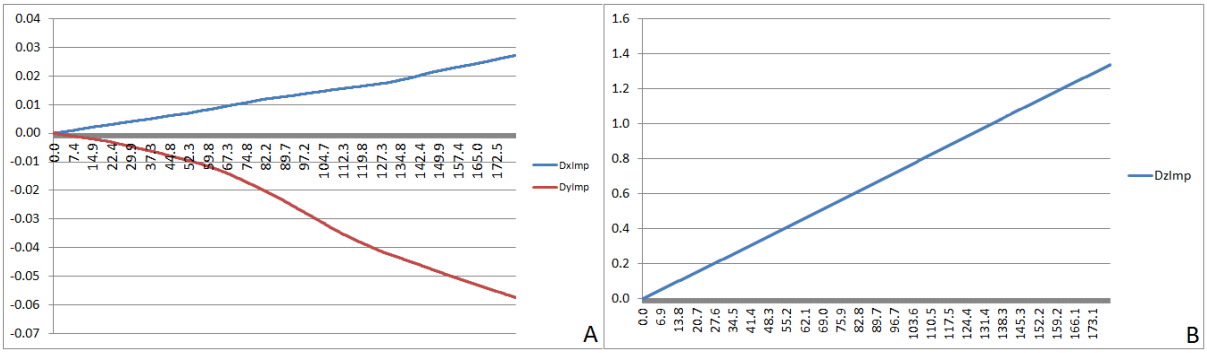

Figure 4.6: Improved displacement graphs in the X and Y directions are shown in graph A whereas the Z direction is plotted in graph B

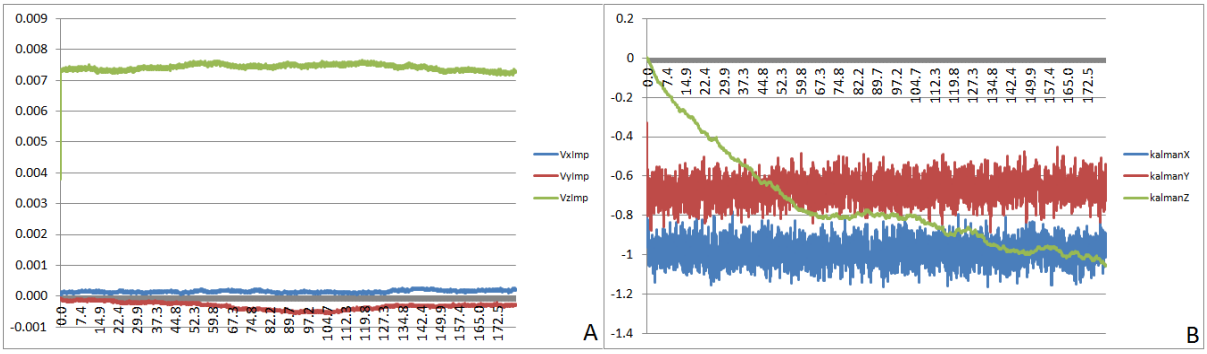

Figure 4.7: A: Improved velocity. B: Kalman filter

Regarding the rotation enhancement methods, the calculated angle after applying Kalman filter is plotted versus time in graph B in Figure 4.7 as shown above. Furthermore, the output from the first and second order complementary filters are plotted below in Figure 4.8 in graphs A and B respectively. Notice the graph similarities between the Kalman and Complementary filters. However, Kalman filter is slightly smoother and better than the Complementary filter in this case. In addition, the second order Complementary filter performs and gives more accurate results than the first order.

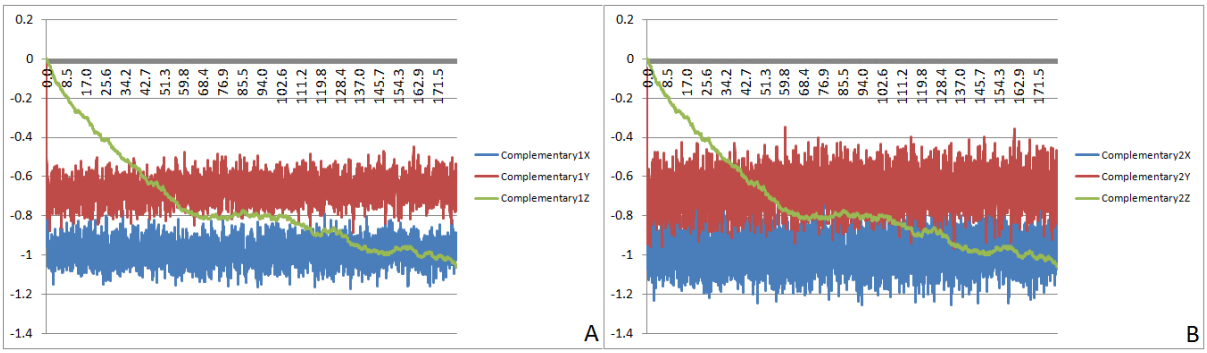

Figure 4.8: 1st and 2nd order Complementary filter

The following two figures show some plotted data after applying the median filter smoothing algorithm which is previously discussed in details in Section 2.4.2. The implemented median filter is based on taking the median value of three readings. Smoothed raw gyroscope data is represented in graph A in Figure 4.9 while graph B shows the accelerometer raw data after smoothing. If you compare graphs A and B in Figure 4.9 with graph A in Figures 4.1 and 4.3 respectively, you will notice that the graphs are smoother and the random fluctuations are removed.

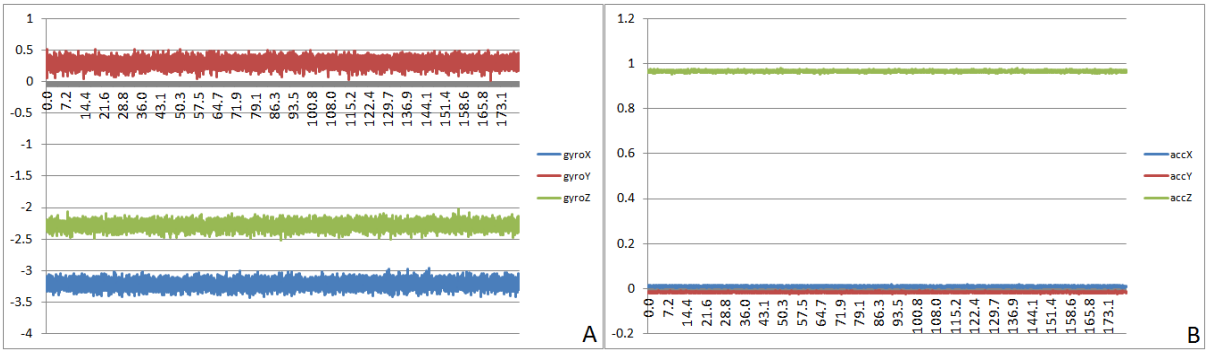

Figure 4.9: Gyroscope and accelerometer graphs after applying the median filter technique

Furthermore, the median filter is integrated with sensor fusion techniques such as Kalman and Complementary filter to give smoothed accurate output. Notice the remarkable improvement and noise elimination when the second order Complementary filter is combined with the median filter as plotted in graph A in Figure 4.19 and compare the result with the one obtained from second order Complementary filter without smoothing in graph B in Figure 4.8. Kalman filter algorithm is enhanced when it is mixed with the median filter as seen in Figure 4.19, graph B. There is a noticeable difference between this graph and the one labeled as B in Figure 4.7.

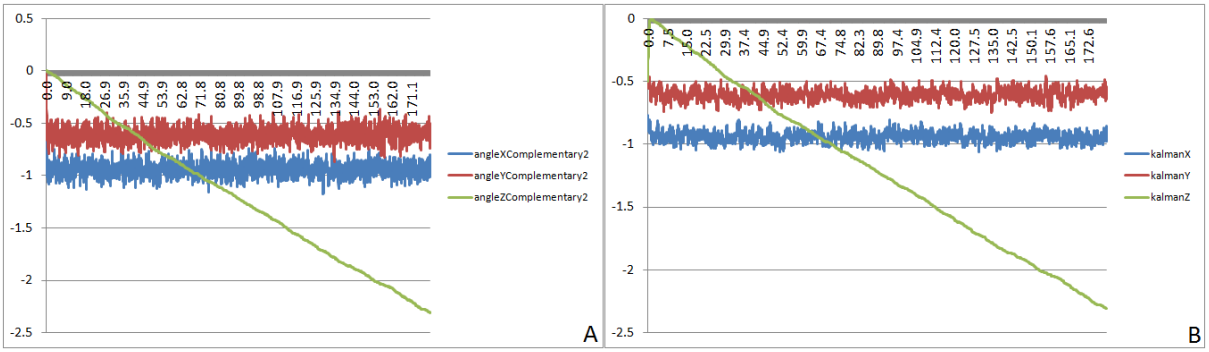

Figure 4.10: Second order Complementary filter (A) and Kalman filter (B) after applying median filter method

Last but not least, Table 4.1 represents a numerical error values comparison between all the algorithms. These error values are obtained by taking data samples at seconds 1 and 61, then, the absolute difference between the two readings is calculated.

Algorithm	Error X	Error Y	Error Z	Average Error
Angle (°)				
Raw Data	193.012	17.066	135.400	115.159
Average Subtraction	1.006	1.347	0.700	1.018
Complementary Filter 1	0.076	0.010	0.749	0.309
Complementary Filter 2	0.114	2.047×10^{-4}	0.749	0.288
Kalman Filter	0.071	0.017	0.748	0.279
Accelerometer Angle	0.012	0.452	0.749	0.404
Velocity (m/s)				
Raw Data	6.746	9.751	569.904	195.467
High Pass Filter	3.781×10^{-5}	1.455×10^{-4}	2.130×10^{-4}	1.321×10^{-4}
Displacement (m)				
Raw Data	210.374	302.264	17667.537	6060.058
High Pass Filter	8.249×10^{-3}	0.012	0.444	0.155

Table 4.1: Different algorithms performance comparison using Arduino project test results

4.1.2 Smartphone Result

The main purpose of this subsection is to explain in detail the result of the experiment when the mobile phone is at rest laying flat for 3 minutes. In fact, the five tests needed to investigate each mode are raw data, average subtraction, median filter applied to average subtraction, average smoothing method with average subtraction, and integrating both smoothing techniques with average subtraction. The results of each test are plotted to compare the performance of the improvement algorithms. Figure 4.11 shows the raw angular velocity values measured from the Android gyroscope sensor (A) versus the angular velocity after applying the average subtraction method (B). All the gyroscope values in graph B are close to zero and that makes sense since the smartphone is not moving.

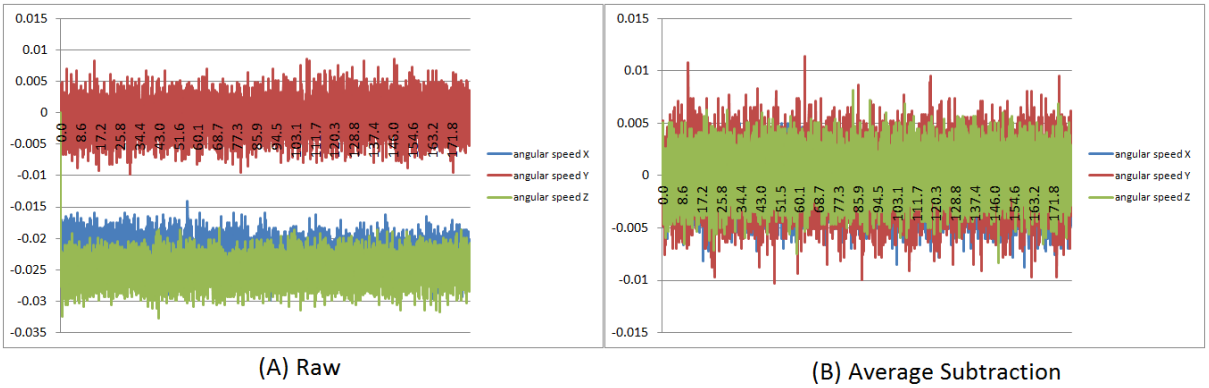

Figure 4.11: A: Raw angular velocity versus time. B: Average subtracted angular velocity

In Figure 4.12, orientation angles, that are calculated from gyroscope only, are plotted against time. Graph A shows the orientation computed from raw angular velocity while graph B illustrates the angles obtained from average subtracted angular velocity. The drift is remarkably diminished due to applying the average subtraction method.

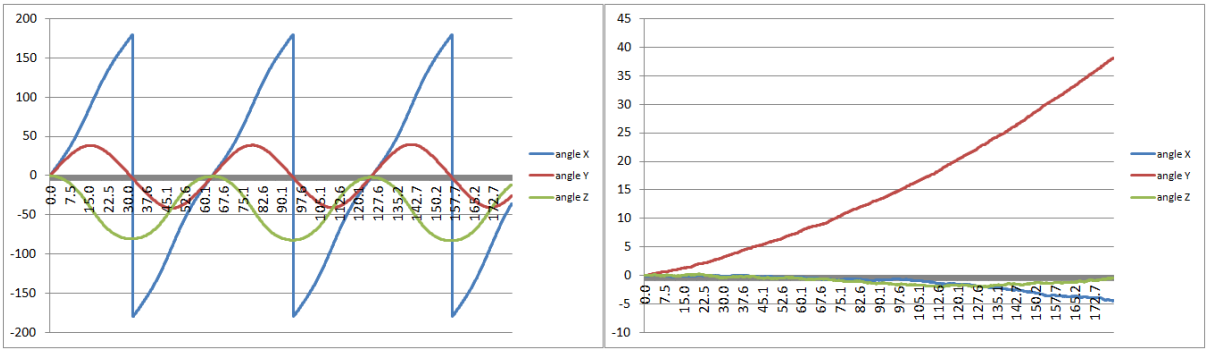

Figure 4.12: Angles from raw gyroscope data and average subtracted angular velocity are plotted in graphs A and B respectively

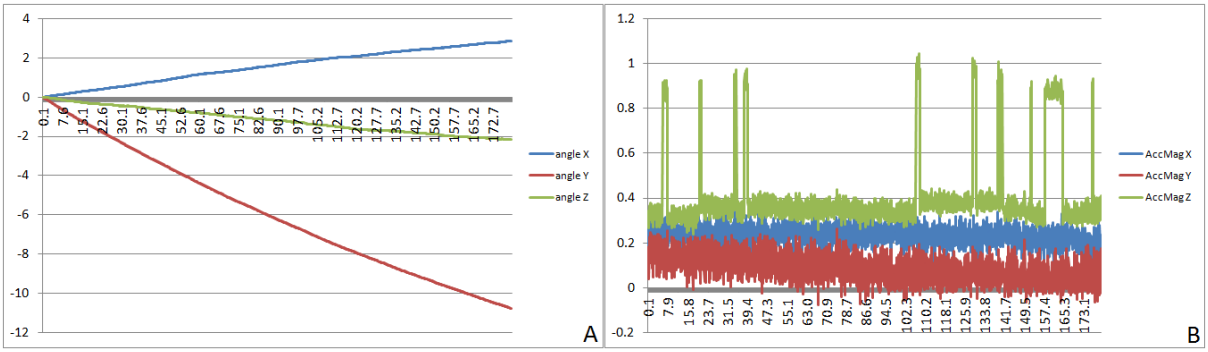

Figure 4.13: A: Angles in all directions are plotted versus time after applying median filter on average subtracted gyroscope data. B: Median filtered accelerometer magnetometer orientation

Another rotation mode in the application is the accelerometer magnetometer orientation such that the azimuth, pitch, and roll are calculated using *TYPE_ACCELEROMETER* and *TYPE_MAGNETIC_FIELD* Android sensors as explained before in Section 3.2.2 in the Methodology chapter. The output of this mode is then used in Kalman and Complementary filters. The difference between the orientation computed from raw accelerometer and magnetometer data (A) versus the average smoothed ones (B) is illustrated in Figure 4.14. The noise in graph A is significantly reduced due to smoothing as shown in graph B. In addition, graph B in Figure 4.13 shows the orientation resulted from applying median filter on the accelerometer and magnetometer sensors.

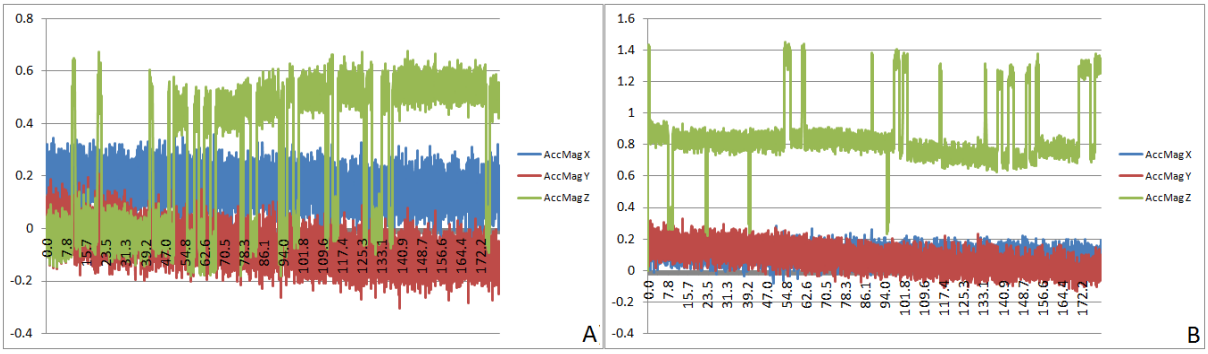

Figure 4.14: A: Raw accelerometer magnetometer orientation. B: Computed angles from average smoothed data of accelerometer and magnetometer sensors

Figures 4.15 and 4.16 illustrate the output orientation from gyroscope and accelerometer sensors combined together using Complementary filter. Graph A in Figure 4.15 shows the angles that are calculated from raw gyroscope data in the X and Z directions while the orientation in the Y direction is plotted in graph B.

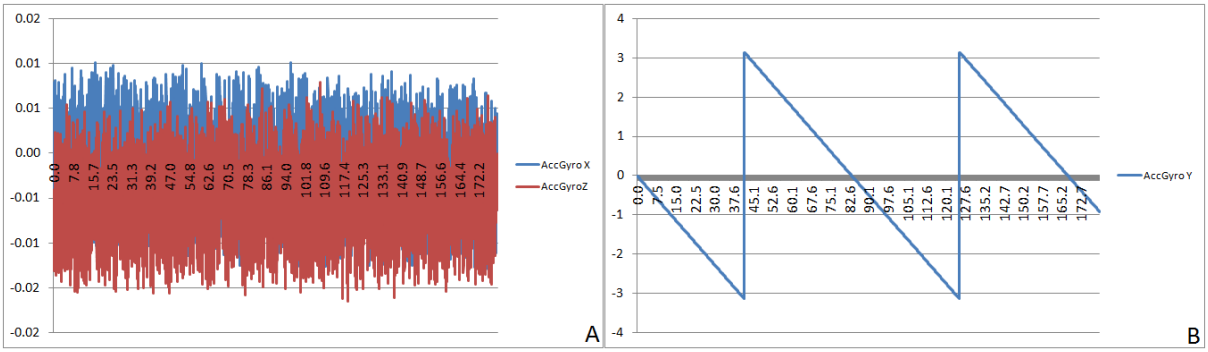

Figure 4.15: Raw gyroscope accelerometer orientation

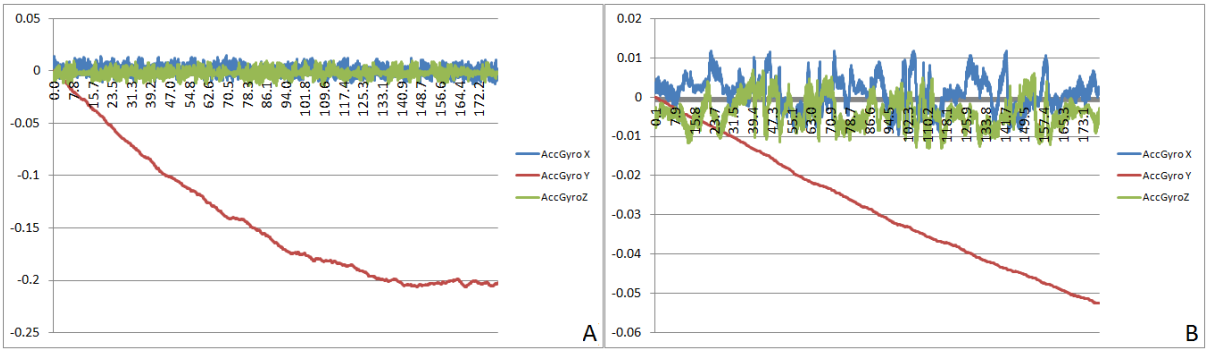

Figure 4.16: Average subtracted gyroscope accelerometer orientation with smoothing method (A) and median filter algorithm (B)

Moreover, the angles computed from average subtracted gyroscope data as well as accelerometer are plotted in Figure 4.16. The difference between the two graphs in this figure is that graph A plots the data after applying average smoothing while graph B shows the result of applying the median filter.

Figure 4.17 contains 4 graphs that demonstrate the Complementary filter performance and how the input data affects the result. Graph A shows the Complementary filter which is applied to raw gyroscope data as well as accelerometer and magnetometer orientation. The other 3 graphs are based on combining the smoothing techniques with the average subtraction method such that the average angular velocity is subtracted from gyroscope data and combined with the angle that is calculated from accelerometer and magnetometer sensors. For example, graph B shows the result of mixing average smoothing and average subtraction whereas graph C illustrates the effect of using the median filter and graph D, which is the last graph in this figure, shows the output of combining all the previous techniques.

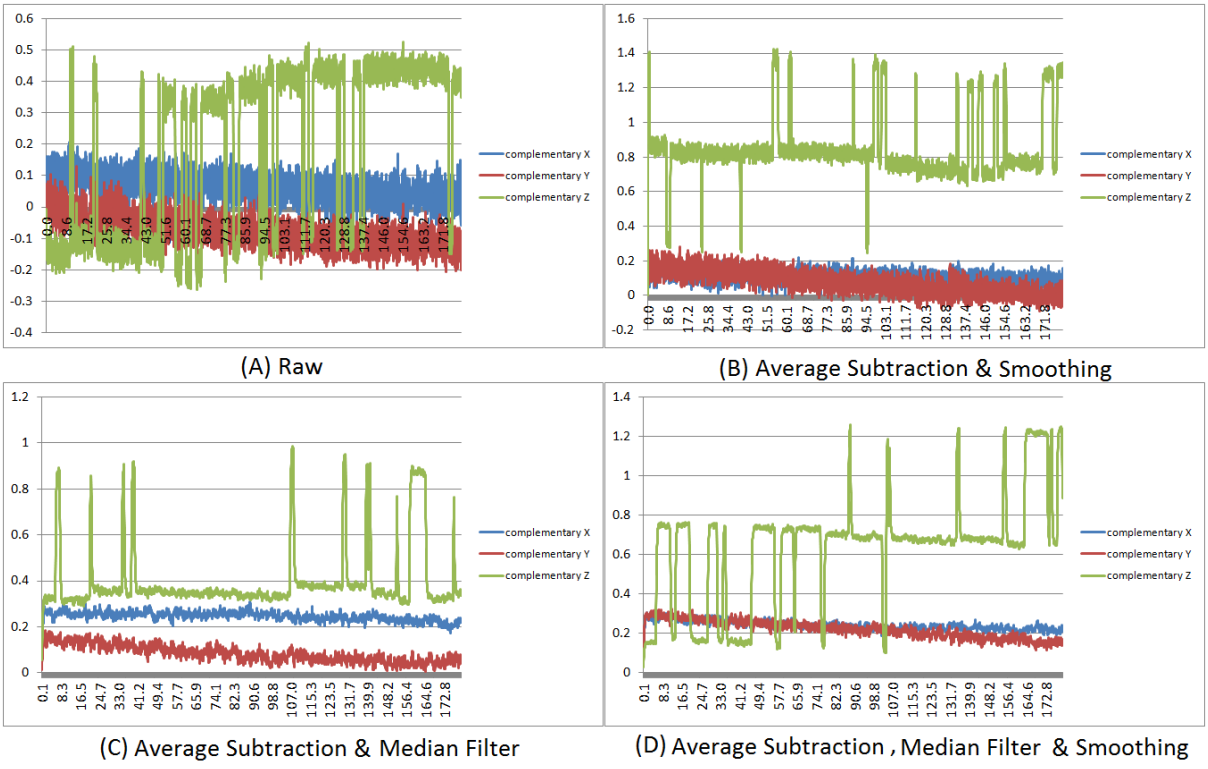

Figure 4.17: Complementary filter performance comparison when used with raw and average subtracted sensors data, and the effect of using average smoothing and median filter is illustrated as well

Figure 4.18 represents the same graphs which are plotted in the previous figure with the same applied methods and graphs order, but Kalman filter is used as a sensor fusion

enhancement technique instead of the Complementary filter. It is clear from the plotted graphs in Figures 4.17 and 4.18 that the average subtracted sensors data outperformed the raw data. Furthermore, the noise is filtered out when smoothing methods are used. According to the plotted graphs, the performance of Complementary and Kalman filters are quite close. However, Kalman filter graphs contain less fluctuations than Complementary filter.

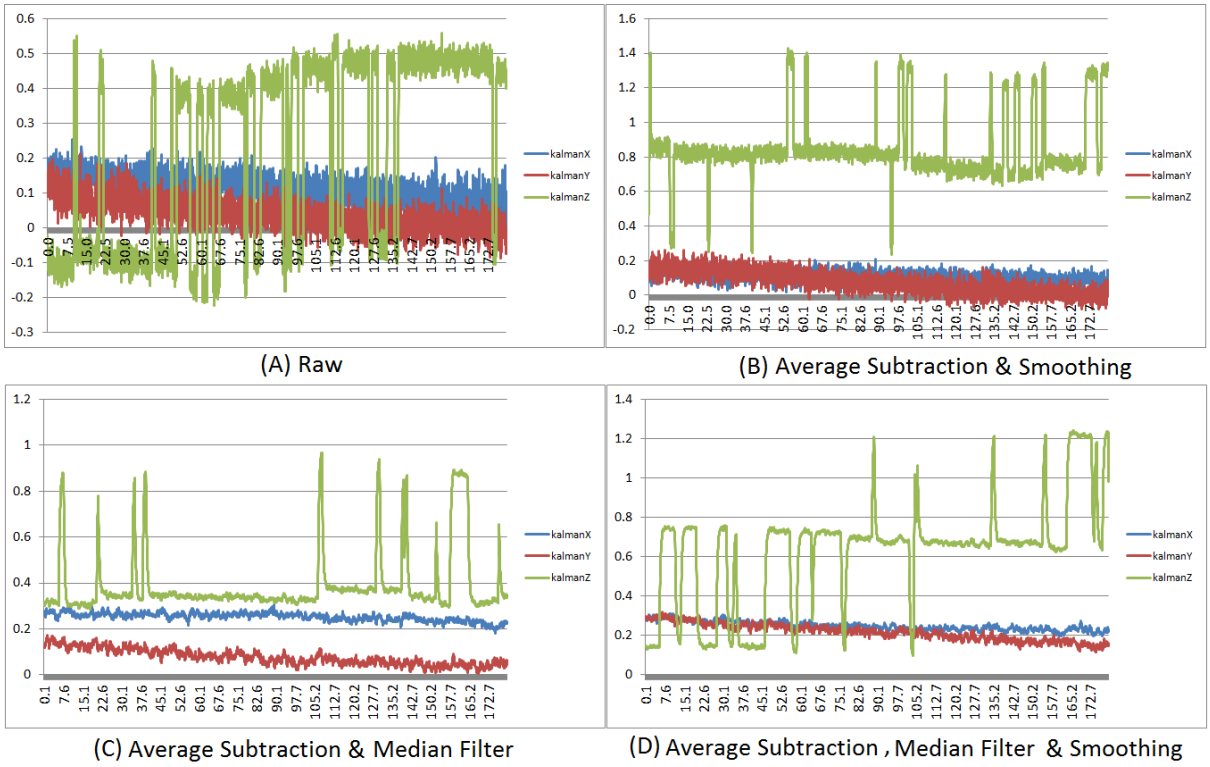

Figure 4.18: This figure is demonstrating the effect of using Kalman filter with a combination of smoothing techniques as well as a comparison of applying the multisensor fusion algorithm on raw and average subtracted sensors data

Last but not least, the orientation angles, which are calculated from the Android virtual sensor *TYPE_ROTATION_VECTOR*, are plotted in Figure 4.19. This sensor is explained in Section 2.1.4 in the Background chapter and it is mainly used to compare its performance with the implemented sensor fusion algorithms which fuse the same hardware sensors data used in the rotation vector sensor. This figure has 4 graphs. The angles, which are computed from the sensor's raw data, are plotted versus time in graph A. Graphs B, C, and D illustrate the output after applying average smoothing method only, median filter only and both respectively. These graphs and the ones resulted from combining smoothing methods with Complementary and Kalman filters are so close indicating that the accuracy levels are almost the same.

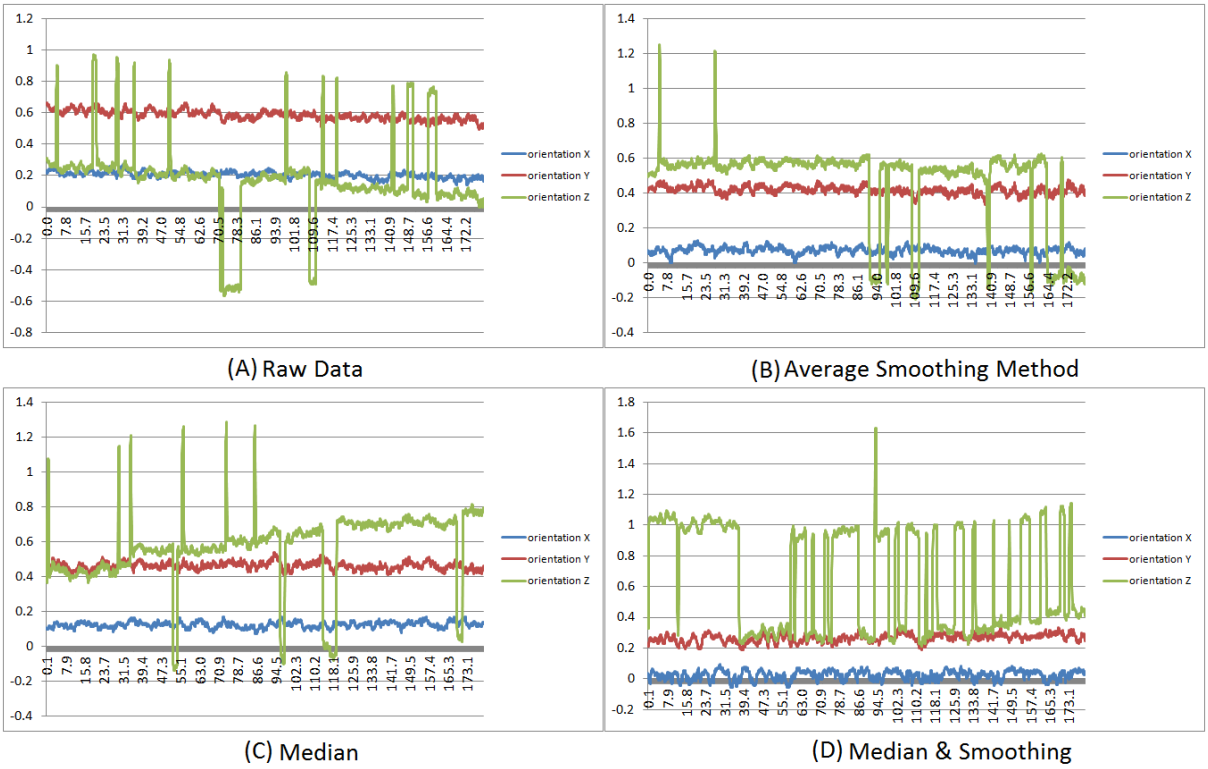

Figure 4.19: Rotation vector sensor orientation

The first translation mode in this application is the raw data mode such that the data measured by the accelerometer is plotted versus time as shown in graph A in Figure 4.20. The second mode is called average subtraction mode, in which the average linear acceleration is subtracted from the data measured by the *TYPE_LINEAR_ACCELERATION* Android sensor, and is represented in graph B in the same figure. This graph shows that the linear acceleration values are more close to zero due to applying the average subtraction method. In addition, graph C shows the result of combining the median filter with average subtraction. Finally, the output of applying the average smoothing method to the previous mode is plotted in graph D.

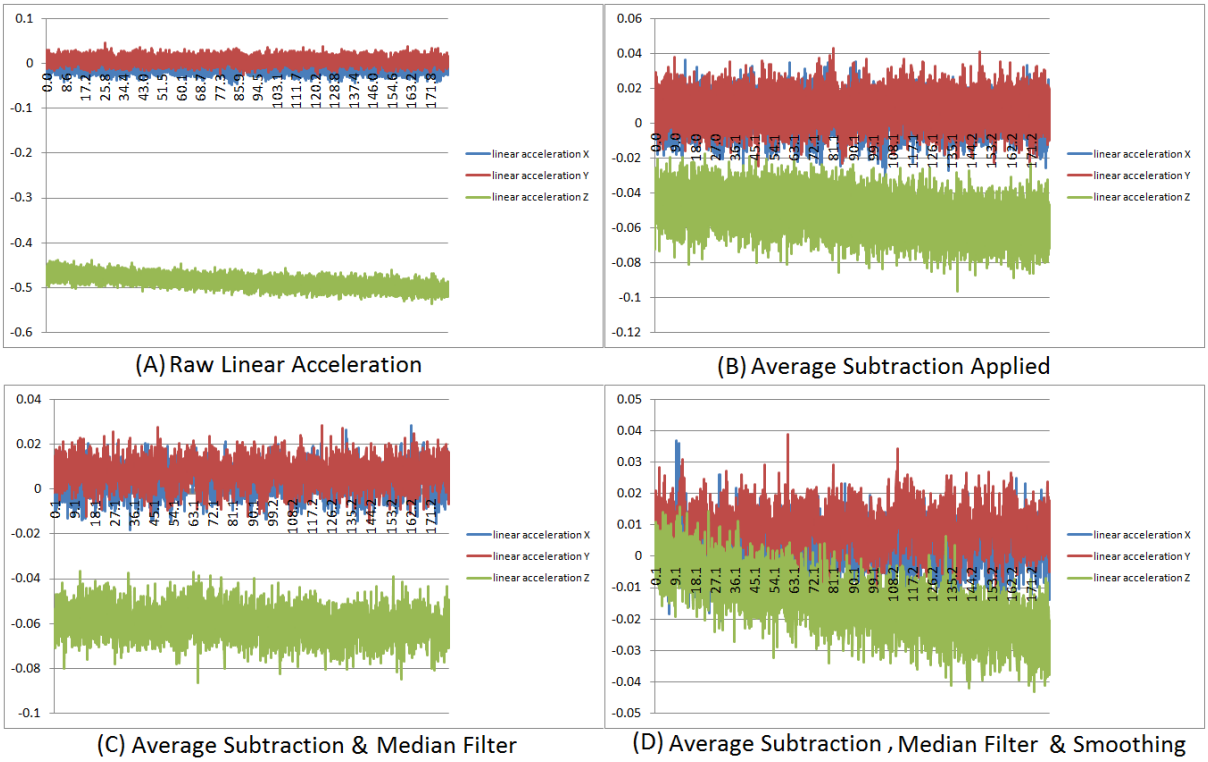

Figure 4.20: Linear acceleration in X, Y, and Z directions are plotted versus time

Figures 4.21 and 4.22 represent the velocity, that is calculated by integrating the raw linear acceleration, and the displacement, which is computed from the raw linear acceleration by double integration, respectively. Velocity and displacement in the x-axis and y-axis are plotted against time in graph A while the Z direction is plotted in graph B. It can be noticed that the velocity and displacement increase with time due to error accumulation although they should be constant and equal to zero as the device is not moving.

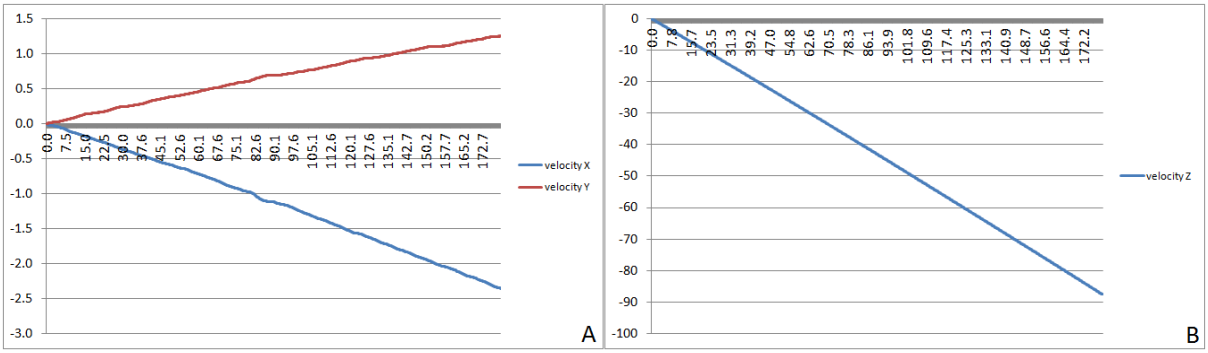

Figure 4.21: Calculated velocity from raw linear acceleration

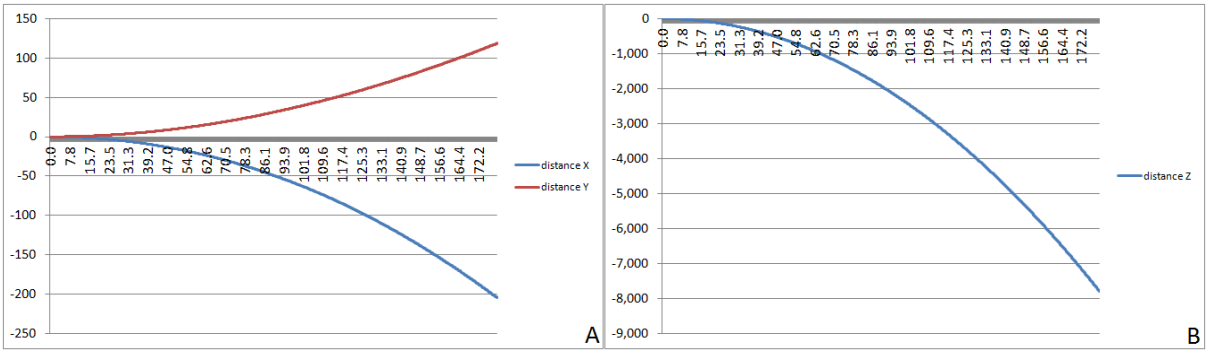

Figure 4.22: Calculated displacement from raw linear acceleration

Figure 4.23 demonstrates the data that are calculated from the linear acceleration after subtracting the average. Graph A shows the velocity (m/s) against time whereas graph B represents displacement, in meters, against time. It can be observed that the error is reduced due to applying this method.

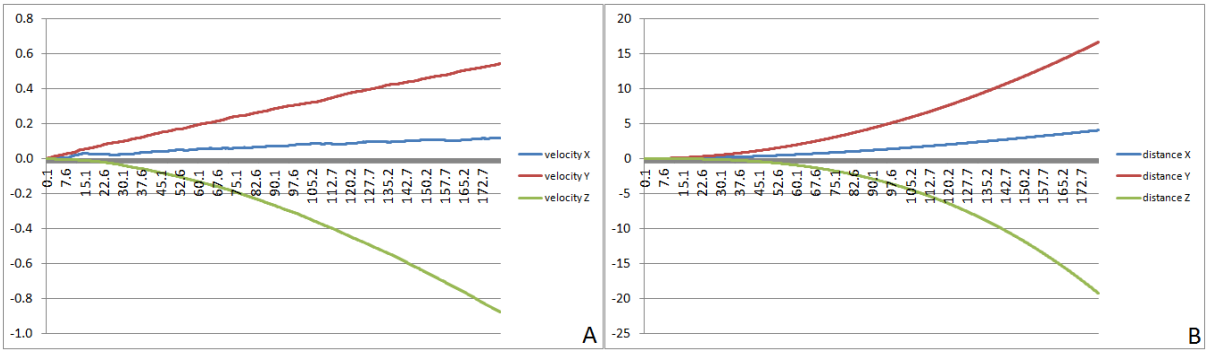

Figure 4.23: Calculated velocity and displacement from average subtracted linear acceleration

Table 4.2 below compares the different translation and rotation modes. The performance and accuracy of multisensor fusion as well as smoothing algorithms is determined by taking a sample of the recorded data at seconds 1 and 61. Then, the error in all directions is calculated and the average error for each method is taken. This average error value is inversely proportional to performance determination such that the smaller the error value, the better the performance of the improvement technique. It can be concluded from the results in the table that average subtraction method significantly reduces the error in rotation angle from 231.536 to 2.992 degrees and in translation velocity from 9.883 to 1.178 m/s and displacement from 304.967 to 36.091 meters. Furthermore, fusing average subtracted gyroscope data with accelerometer orientation using Complementary filter outperformed the orientation that resulted from integrating of accelerometer and magnetometer data. Despite the fact that Kalman filter showed better performance than

the Complementary filter, it did not outperform *TYPE_ROTATION_VECTOR* sensor fusion which is provided by Android.

Algorithm	Error X	Error Y	Error Z	Average Error
Angle (°)				
Raw Data	345.632	167.404	181.572	231.536
Average Subtraction	0.308	8.031	0.636	2.992
Complementary Filter 1	0.037	0.083	0.574	0.231
Kalman Filter	0.035	0.071	0.566	0.224
Accelerometer Magnetometer Orientation	0.069	0.055	0.576	0.233
Accelerometer Gyroscope Orientation	6.180×10^{-3}	5.810×10^{-3}	9.260×10^{-3}	7.083×10^{-3}
Rotation Vector Sensor	0.023	0.057	0.104	0.061
Velocity (m/s)				
Raw Data	0.708	0.458	28.482	9.883
Average Subtraction	0.248	0.481	2.806	1.178
Displacement (m)				
Raw Data	22.191	14.371	878.399	304.967
Average Subtraction	7.512	14.966	85.795	36.091

Table 4.2: Comparing different techniques implemented in Android application

4.2 Motion Test

Multisensor fusion algorithms, that are implemented in the project, are tested to investigate their accuracy level. This is done by fixing the mobile phone in a certain position such that its back is on the table. Then, the smartphone is moved and rotated in certain known directions and returned back to its initial position. A graph for average subtracted gyroscope orientation as well as Complementary and Kalman filters angles are plotted for each experiment. The final Kalman or Complementary filter orientation value is almost the same as the initial value when the device is returned back to its original position. Nevertheless, the gyroscope angles drift with time and the orientation of the device does not return back to its original value even when the motion test is finished. The first motion test is rotating the device randomly in all directions and plotting the results. Figure 4.24 is showing the orientation of Complementary and Kalman filters in the X, Y, and Z directions in graphs A and B respectively while graph A in Figure 4.25 represents the output gyroscope angles as a result. It can be observed from these graphs that fusing all the hardware sensors data using Kalman and Complementary filters outperformed the result of improved gyroscope.

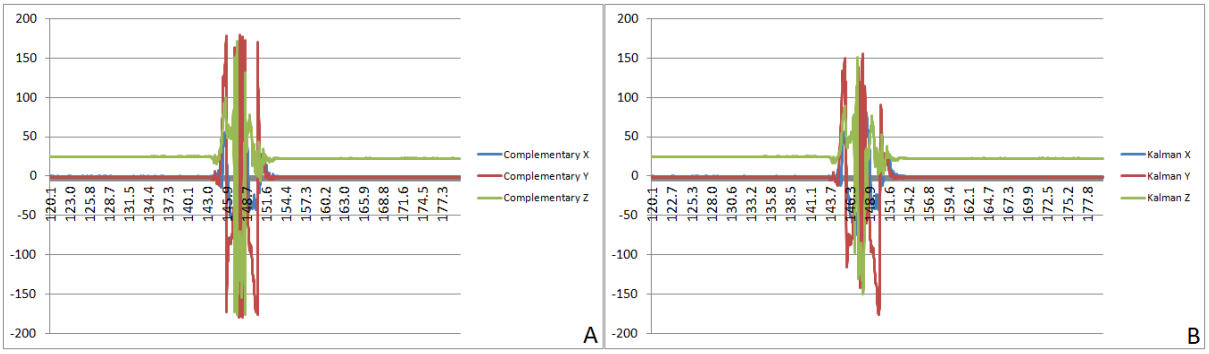

Figure 4.24: Complementary and Kalman filters graph for the random motion test

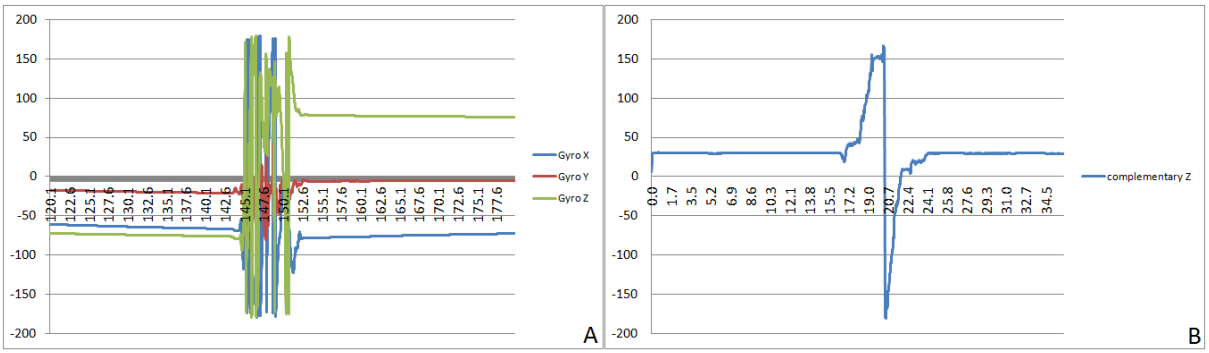

Figure 4.25: A: Angles calculated from average subtracted angular velocity. B: The resulted orientation of Complementary filter when the device is rotated around the z-axis

The second motion test is rotating the device around the z-axis only. Graph B in Figure 4.25 as well as graphs A and B in Figure 4.26 show the results of the Complementary filter, Kalman filter, and gyroscope orientation respectively during this motion.

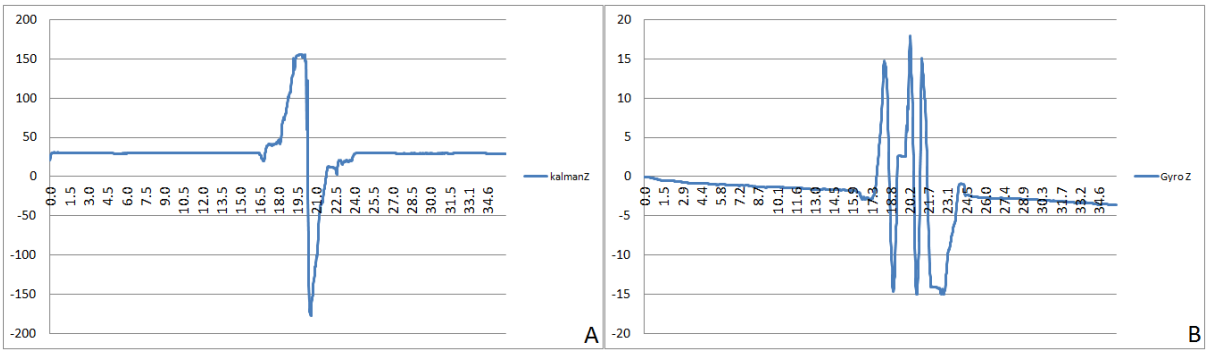

Figure 4.26: This graph represents the z-axis rotation output orientation of Kalman filter and average subtracted gyroscope values

The previous test is repeated, but the rotation is around the x-axis. In Figure 4.27 shown

below, the resulted x-axis orientation of Complementary and Kalman filters are plotted while the output gyroscope angles are plotted against time in graph A in Figure 4.28.

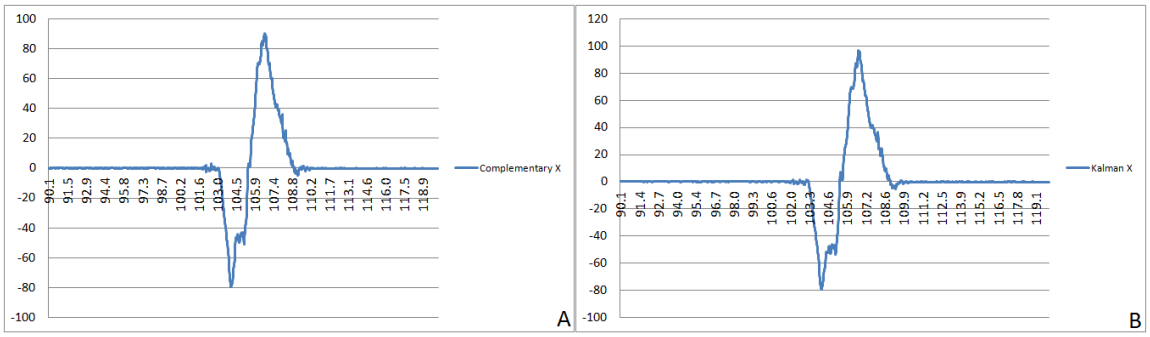

Figure 4.27: Complementary and Kalman filters graph for x-axis rotation test

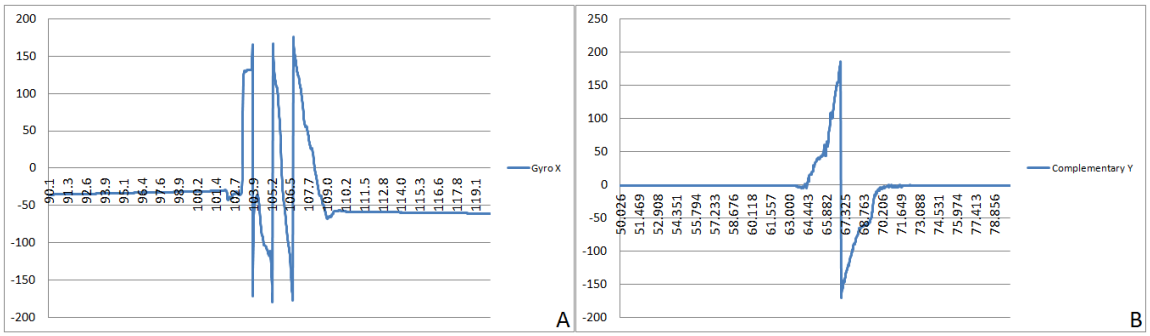

Figure 4.28: A: Calculated angle from improved gyroscope data around x-axis. B: The resulted Complementary filter orientation when the device is rotated around the y-axis

The last test is rotating the device around the y-axis the same way as the previous two tests. Complementary filter result is illustrated in graph B in Figure 4.28 while Kalman filter and gyroscope orientation around y-axis are plotted versus time in Figure 4.29.

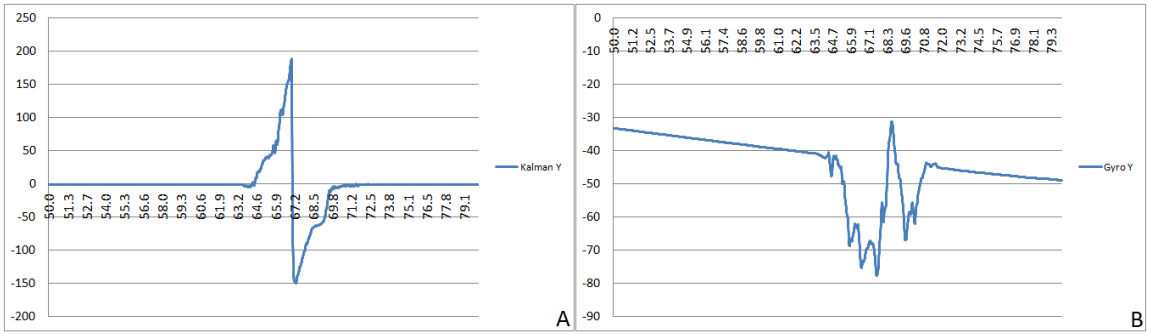

Figure 4.29: Typical y-axis graph representation of the Kalman filter output roll (A) as well as gyroscope orientation (B)

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The main objective of this thesis is to examine the accuracy level of sensor fusion that can be achieved in Android devices and embedded systems. One of the most common applications of this project is indoor positioning which has grasped great attention in recent years. This project is mainly focused in fusing data from several built-in sensors, that are available in cellular phones like accelerometers, gyroscopes, and 3D compasses, to enhance translation and orientation accuracy. In fact, the project is based on hardware Arduino sensors in addition to smartphone sensors. It consists two independent parts; a mobile application, which is implemented in Java on the Google Android Operating System (OS), and an Arduino project that uses a 3D gyroscope and accelerometer module called MPU-6050 and is implemented using Arduino and Processing IDE. Several sensor fusion and smoothing algorithms such as median filter, first and second order Complementary filters as well as Kalman filter are implemented in both applications. These improvement algorithms, as well as the mobile phone and hardware sensors quality and efficiency, are evaluated and compared. Stationary and motion tests are the two ways used to investigate the sensor fusion and noise filtering methods. The phone or the external sensor is immobile in the stationary test while in the motion experiment, the device is rotated around one or more axes and returned back to its original position.

According to the computed results, the average errors of the orientation, that is calculated from raw and average subtracted sensors data, are 115.159 and 1.018 degrees in the MPU-6050 sensor whereas the average errors in phone's sensors are 231.536 and 2.992 degrees. Besides, it can be observed that the error in velocity is reduced from 9.883 to 1.178 m/s and in displacement from 304.967 to 36.091 meters when the average subtraction method is applied to the sensors that are placed in smartphones. On the other hand, the average error in velocity and displacement are noticeably diminished from 195.467 to 1.321×10^{-4} m/s and from 6060.058 to 0.155 meters respectively due to passing raw acceleration, that is measured by the accelerometer in MPU-6050, over a high pass filter

to eliminate the gravitational force component as well as noisy signals. Concerning sensor fusion experimental results, Kalman filter showed an outstanding performance such that the average errors are 0.279 and 0.224 degrees in the MPU-6050 and Android sensors respectively. Moreover, Complementary filter accuracy was quite close to Kalman. The average errors, that resulted from applying the first order Complementary filter on external hardware Arduino and built-in Android sensors, are equal to 0.309 and 0.231 degrees respectively. Furthermore, the average smoothing algorithm as well as median filter significantly reduced the noise and removed the spikes and fluctuations that appeared in the plotted graphs. In conclusion, the level of accuracy was quite satisfactory and sensor fusion techniques significantly enhanced the sensors' performance and gave more accurate results than the output from individual sensors.

5.2 Future Work

A number of enhancement ideas can be added for further development.

- Increasing the accuracy of the translation like velocity and displacement calculation is identified as the most essential. This could be done by using Kalman filter for translation enhancement.
- Combine more than one accelerometer, apply filters on them and fuse their readings to get more accurate results.
- More advanced algorithms could be implemented for further rotation improvement such as extended Kalman filter.
- Incorporate sensor fusion techniques in some smartphone and embedded system applications like virtual and augmented reality applications, interactive games, Human Computer Interaction (HCI) devices and biomedical applications.

Appendix

Appendix A

Lists

rpm	revolutions per minute
CSV	Comma Separated Value
IDE	Integrated Development Environment
DOF	Degrees of Freedom
IoT	Internet of Things
I/O	Input/Output
PDE	Processing Development Environment
I2C	Inter-Integrated Circuit
SDA	Serial Data
SCL	Serial Clock
LQE	Linear Quadratic Estimation
PC	Personal Computer
NFC	Near Field Communication
USB	Universal Serial Bus
EEPROM	Electrically Erasable Programmable Read-Only Memory
KB	Kilo Byte
SRAM	Static Random Access Memory
μ**T**	micro Tesla

GPS	Global Positioning System
OS	Operating System
HCI	Human Computer Interaction
DoD	Department of Defense
MEMS	Micro Electro-Mechanical System

List of Figures

List of Tables

Bibliography

[1] Applying Low Pass Filter to Android Sensor's Readings. https://www.built.io/blog/applying-low-pass-filter-to-android-sensor-s-readings. (Accessed on May 2017).

[2] Arduino UNO and Genuino UNO. https://www.arduino.cc/en/main/arduinoBoardUno. (Accessed on May 2017).

[3] Develop, API Guides, Location and Sensors, Motion Sensors. https://developer.android.com/guide/topics/sensors/sensors_motion.html. (Accessed on May 2017).

[4] Kalman filter vs Complementary filter. http://robottini.altervista.org/kalman-filter-vs-complementary-filter. (Accessed on May 2017).

[5] Multisensor fusion and integration: Approaches, applications, and future research directions.

[6] Processing. https://processing.org/reference/environment/. (Accessed on May 2017).

[7] Quaternion Math. Quaternion based attitude representation. Application Note, AN002.

[8] Samsung Galaxy S4. http://www.samsung.com/uk/smartphones/galaxy-s4-i9505/GT-I9505ZKABTU/. (Accessed on May 2017).

[9] SensorEvent. https://developer.android.com/reference/android/hardware/SensorEvent.html. (Accessed on May 2017).

[10] Understanding Quaternions. http://www.chrobotics.com/library/understanding-quaternions. (Accessed on May 2017).

[11] Shane C. The Balance Filter. June 25, 2007. Rev. 1.

[12] Kushner D. The Making of Arduino. http://spectrum.ieee.org/geek-life/hands-on/the-making-of-arduino, October 26, 2011. (Accessed on May 2017).

[13] Manjoo F. A Murky Road Ahead for Android, Despite Market Dominance. https://www.nytimes.com/2015/05/28/technology/personaltech/a-murky-road-ahead-for-android-despite-market-dominance.html?_r=0, May 27, 2015. (Accessed on May 2017).

[14] InvenSense. *MPU-6000 and MPU-6050 Product Specification*, October 24, 2011. PS-MPU-6000A-00. Rev. 3.1.

[15] Sloth K. A practical approach to Kalman filter and how to implement it. http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-to-implement-it/, September 10th, 2012. (Accessed on May 2017).

[16] David L. and James L. An Introduction to Multisensor Data Fusion. January 1997.

[17] Gabriel A. Terejanu. Discrete Kalman Filter Tutorial. August 4, 2012.

[18] Shala U. and Rodriguez A. Indoor Positioning using Sensor-fusion in Android Devices. September 2011.

[19] Elmenreich W. An Introduction to Sensor Fusion. November 19, 2002.

[20] Zhu Y. and Huang C. An Improved Median Filtering Algorithm for Image Noise Reduction. 2012.

[21] Pavol K. ubica I. and Alojz K. Pedestrian Indoor Positioning and Tracking using Smartphone Sensors, Step Detection and Map Matching Algorithm. 2016.