Orientation Determination for Android Smartphones

Peter Wozniak[1], Katharina Mehner-Heindl[1]

Fakultät Medien und Informationswesen, Hochschule Offenburg, Badstraße 24, 77562 Offenburg, Germany[1]

Abstract

Many Virtual Reality, Augmented Reality, gesture based applications or games on smartphones or tablets require the tracking of the orientation of the device. This is achieved by using built-in sensors such as accelerometer, compass and gyrometer. By using sensor fusion algorithms the accumulated errors of individual sensors can be corrected and the tracking quality can be improved. We propose an algorithm and a multithreaded architecture for Android smartphones that interpolates between the orientation gained from accelerometer and compass and the rotation gained from the gyrometer. Rotations are presented using quaternions. We illustrate the solution with a simple 3D simulation. We compare our solution with the Android built-in virtual rotation sensor.

1 Introduction

The information about the current orientation of a handheld device can be used in various applications. A widely known use case is the automatic screen rotation feature of the operating system on a smartphone or tablet device, which changes the orientation of the screen content based on the inclination of the device. For instance, if the device is held almost upright and tilted either to the left or to the right for more than 45°, the screen content is turned 90° in the opposite direction. This feature is accomplished with an accelerometer sensor. Besides this built-in functionality, one can develop own applications that report fine grained changes in inclination. The readings of an accelerometer e.g. can be used for a virtual spirit level, a tilt-sensitive game controller or a motion detector.

To enhance the orientation determination, it is possible to additionally use a compass and calculate device rotation around the gravitational vector. Thus it becomes possible to describe the device orientation in relation to the earth. This orientation can be used for interactive virtual and augmented reality applications like e.g. Google Sky Map, Wikitude or Google Cardboard virtual reality apps.

This paper is organized as follows. In section 2 we describe sensor characteristics in more detail. In section 3 we detail our requirements for orientation determination. In 4 we present our algorithm and its multithreaded architecture. Here, we also provide an initial evaluation. In 5 we discuss related work and in 6 we present our conclusions.

2 Sensors for orientation determination

The built-in sensors of mobile devices consist of three independent orthogonally arranged sensors, which allow three-dimensional spatial sensor readings of accelerations and the surrounding magnetic field. The resulting readings can be interpreted as a three-dimensional vector describing a spatial direction with respect to the surface of the earth and the north pole.

The quality of the sensor readings is crucial for the outcome of the orientation determination. Due to physical and economical restrictions sensors of current mobile devices are non-perfect. The installed accelerometer sensors consist of very small structures and tend to take noisy measurements, which makes it impossible to distinguish between actual accelerations and random noise. Furthermore, the acceleration readings result out of a combination of different applied forces, which makes it impossible to distinguish between the gravitational and additional linear acceleration. The gravitational vector cannot be deduced. The readings of the digital magnetometer sensor are subject to noise and to local magnetic disturbances, which lead to a deviated North Pole location.

To improve the orientation determination it is necessary to filter the raw sensor data. A simple method of filtering raw sensor readings is to subsequently calculate the moving average of a defined number of last measurements. This low-pass filter implementation reduces noise artifacts and high frequency signal components and thus smoothes the signal reading. While being fully viable for many applications, this approach introduces a negative impact in form of a delay, where the filtered measurement lags behind the actual sensor reading. On the contrary, especially interactive applications like games or VR benefit from a timely and accurate measurement. For these a less lagging filtering approach is needed.

More sophisticated mobile handheld devices often include a supplementary gyroscope sensor, which can be used to improve the orientation determination by providing an independent additional measurement. The gyroscope sensor lacks the possibility of absolute orientation determination and only measures the rotational movement. A gyroscope sensor makes it possible to measure the rotational velocity of the device very precisely, without being negatively affected by further accelerative forces or magnetic disturbances. It is this characteristic in particular, that allows comparing it with the accelerometer sensor readings and concluding on the shares of gravitational and further accelerations. On the other hand, any gyrometer is subject to sensor noise and drift, i.e. an accumulated deviation over time.

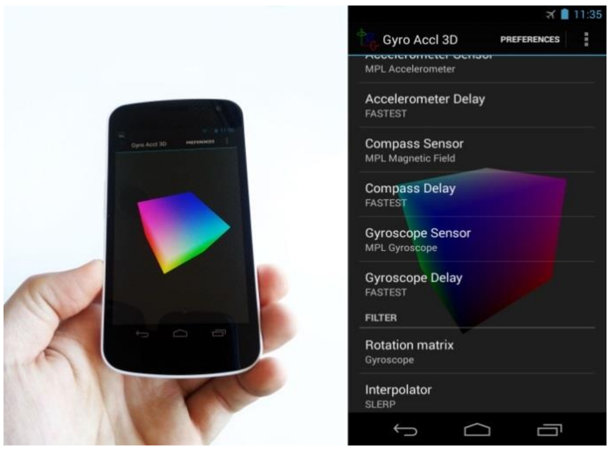
The comparison and interpretation of various sensor measurements in order to gain one joint result is known as sensor fusion, see e.g. (Chen et al. 1999). There are different sensor fusion approaches with various strengths and weaknesses resulting in algorithms with various degrees of complexity. It is therefore all the more important to know exactly the requirements.

3 Requirements

The goal of our algorithm is to determine the orientation of a handheld device in relation to earth by using built-in sensors. The orientation determination shall not suffer from the above mentioned individual sensor insufficiencies. To achieve an orientation as precise as possible, the algorithm will use the accelerometer sensor to measure inclination towards the gravitational vector and the magnetic sensor to measure rotation around the gravitational vector. The gyroscope sensor is used to capture independently the devices’ rotary movement.

The orientation determination should be timely, as many VR/AR applications or e.g. games require a delay as short as possible. It is also expected that the orientation changes are smooth, i.e. the ultimate rendering should not be shaky or wobbly. Here, shaky describes that the orientation is subject to sensor noise, i.e., tiny rotations at a high frequency in any direction are measured, and, unless filtered, are visible in the user interface. Wobbly refers to situations where accelerations un-intuitively and unwantedly contribute to orientation determination, i.e., unless filtered, a deviating orientation at a lower frequency is observed and would become visible in the user interface. However, this deviation is only observable for the period of acceleration. To a certain extent this can be avoided by the already mentioned filtering with the tradeoff of lagging behind.

In order to visualize the calculated orientation and to evaluate its quality, we envisioned a simple prototype application. A 3D cube (cf. figure 1) will be drawn on the screen and animated contrary to the device rotation. This gives the impression that the cube has a fixed spatial position. Rotations around the z-axis will not result in viewing new parts of the cube, only in rotation of the view, while rotations around x- or y-axis show new parts of the cube.



MPL Accelerometer
Accelerometer Delay
FASTEST
Compass Sensor
MPL Magnetic Field
Compass Delay
FASTEST
Gyroscope Sensor
MPL Gyroscope
Gyroscope Delay
FASTEST
FILTER
Rotation matrix
Gyroscope
Interpolator
SLERP

Figure 1 Running Android prototype application and its settings menu

The sensor fusion algorithm requires Android 2.3 (as this is the first Android version supporting gyroscope sensors) and newer. The mobile device at least needs the above mentioned sensor types, an acceleration sensor, a compass sensor and a gyroscope sensor. In order to guarantee a fluid user experience, according to the Android SDK, long lasting calculations should be performed without blocking the main application thread. Therefore it is desirable to design the algorithm efficiently and take advantage of multicore CPUs by performing calculations concurrently.

4 Architecture Design

The following part describes our approach for a multithreaded Android based sensor fusion software architecture. The focus of this project was on the demonstration of a concurrent, responsive sensor fusion implementation that can be reused for various Android applications requiring orientation determination.

4.1 Quaternion-based complementary filter

We decided to use a sensor fusion approach that works with a quaternion interpolation algorithm, whose concept is derived from Shane Coltons balance filter (Colton 2007). His complementary filter based approach is a weighted interpolation between two independently calculated states representing the inclination of a device. In opposition to his two-dimensional approach our implementation uses quaternions to represent the calculated orientations. The first quaternion is calculated using the acceleration and magnetometer sensor data and therefore represent the devices orientation in relation to earth (cf. figure 2). An Android helper method is used to calculate the corresponding rotation matrix, which afterwards is converted into a quaternion representation. The second calculated quaternion results from the gyroscope sensor data und thus only represents the devices rotation performed since the beginning of measurements. Furthermore they both inherit the characteristic advantages and disadvantages of their respective sensors.

The orientation represented by the first quaternion is shaky and wobbly, like the underlying accelerometer and magnetometer sensor readings are. The orientation represented by the second quaternion is very responsive and immune to these measurement errors. In principle, for every measurement interval the measured angular velocity must be multiplied with the interval’s duration to get the covered rotational angle. All calculated intermediate rotations are added up to an initial orientation state and in the end represent the new updated orientation of the device. As the gyroscope sensor’s readings are never perfectly exact, the numerous inaccuracies over time sum up to a growing deviation called sensor drift and therefore to an erroneous orientation determination.

During the initialization of the algorithm the second quaternion, based on the gyroscope sensor data, is equated with the first one, so that both represent exactly the same device orientation gathered by accelerometer and magnetometer sensor measurements. During runtime after every sensor reading interval, an update and recalculation of the corresponding quaternions is taking place.

An interpolation between these two quaternions is the key to successfully filter out the unwanted disadvantages and emphasize the advantages from both measurements. In our case we want a sensor fusion algorithm that provides a non-drifting, non-shaky and non-lagging orientation determination of our handheld device. A spherical linear interpolation (SLERP) between the two quaternions allows us to weight between the two orientation estimations we have and in particular between their specific characteristics (Shoemake 1985). As long as both quaternions represent an absolute perfect and flawless orientation, the interpolation result will be identical to them. But in reality disturbances and inaccuracies will drive both estimations apart.

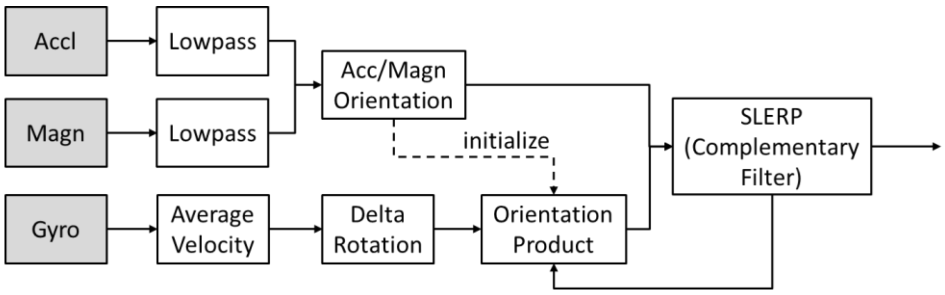


Figure 1 The quaternion-based complementary filter structure

Weighting the gyroscope sensor data based orientation with 98% or more and the accelerometer/magnetometer based one with less than 2%, produces a very smooth and responsive orientation estimation without a lag, shakiness and drift. As the gyroscope produces a very responsive and accurate estimation for short terms, its calculation is rated very high. To compensate for the gyroscope sensors drift a small amount of the interpolation goes towards the estimation coming from the accelerometer and magnetometer sensors, which are more exact in the long term. In order to prevent the gyroscope based orientation to drift apart, it is necessary to equate and thus correct the orientation product with the interpolated solution after each interval.

4.2 Concurrent sensor fusion

Besides developing the sensor fusion algorithm itself, it was necessary to take care of including it within Androids software framework. One main strategy for developing Android applications is the reduction of load within the main application thread, in order to improve responsiveness and user experience. As modern mobile devices are equipped with multicore CPUs, it became evident to take advantage of them by designing our sensor fusion as a concurrent software thread. The goal was to shift most of the calculations regarding the sensor fusion into a dedicated thread and reduce the overhead produced by the garbage collector by reusing as many data structures as possible, thereby giving the app a better memory footprint. The latter point proved to be quite important as the sensors read-out rate can easily reach up to 100Hz.

The sensor fusion thread’s live cycle has been linked to the life cycle of the main applications Activity. Thus the initialization, starting and stopping of the sensor fusion itself is synchronized with the application. Android’s software framework requires the usage of an operating system service called SensorManager to get access to the devices sensors. It is necessary to register for the desired sensors and to implement a callback method named onSensorChanged() (Milette & Stroud 2012). This method sorts the incoming sensor data according to its origin, briefly preprocesses it and then stores the measurements in data structures that are shared with the sensor fusion thread. As this method runs in the scope of the main applications thread, it is necessary to synchronize the access to some of the shared data structures. We decided to limit the amount of synchronization to a minimum and thus only control the access of the gyroscope sensor data. As a matter of fact the gyroscope sensor data is crucial for a correct orientation determination. Firstly the chosen weighting overemphasizes its impact on the orientation. Secondly it is vital to capture all intermediate measurements from the gyroscope sensor, to get a full summation of the measured rotations. For each sensor reading of the gyrometer that occurs, we compute the average velocity between the actual reading and the average velocity of a series of previous readings. The average velocity (also known as mean or effective velocity), is an arithmethic average weighted over the time intervals between the time stamps of subsequent readings. Conveniently, it can be computed incrementally, i.e. computing the average velocity from n readings A_n is the same as computing from n-1 readings A_{n-1} and then computing the weighted arithmetic average between A_{n-1} and one new reading.

A rotation matrix is only computed, when the sensor fusion thread has new accumulated sensor data. The frequency of the fusion thread therefore is tied to the sensors rate. Also in our sample app it is parameterizable at which rates the sensors are working and if and how long the fusion thread may sleep per cycle. Furthermore regarding synchronization, it has no noticeable impacts on the orientation determination if the access on the shared data structures of the accelerometer or magnetometer sensor isn’t synchronized.

4.3 Comparison

It is eligible to compare the performance of our orientation determination with Android’s synthetic Rotation Vector sensor. In spite of the fact, that both offer virtually the same functionality, it is very difficult to draw general conclusions. It is up to the manufacturer of a device on how they implement the Rotation Vector functionality. It could be a very simple software-based solution, but it could also include a gyrometer-based filtering or it could even be a hardware-based Motion Processing Unit (MPU) offering the functionality. Every Android device is slightly different and thus offers a slightly different performance. Even different software revisions on identical devices could result in a change of the performance of the Rotation Vector. E.g. the Rotation Vector sensor of a Samsung Galaxy S3 (i9300) with Android 4.3 stock firmware offers a very good orientation determination of the device. The Galaxy S3 is equipped with an Invensense MPU-6050 MEMS sensor and a motion processing processor, which offers optimized embedded sensor fusion processing. While running the same device with an alternative custom Android 4.4 firmware from Cyanogenmod, the Rotation Vector is less accurate and timely.

For a better comparison and analysability we converted our calculated quaternions into Euler angels and plotted them. Figure 2 shows a small section of the measured y-axis orientation of all involved quaternions for a steady and an agitated device. It is easy recognizable that the purely gyroscope-based quaternion is drifting while the low-pass filtered accelerometer- and magnet-sensor-based quaternion is not drifting over time. While shaking the device the accelerometer- and magnet-sensor-based orientation gets heavily disturbed due to the additional accelerative forces.

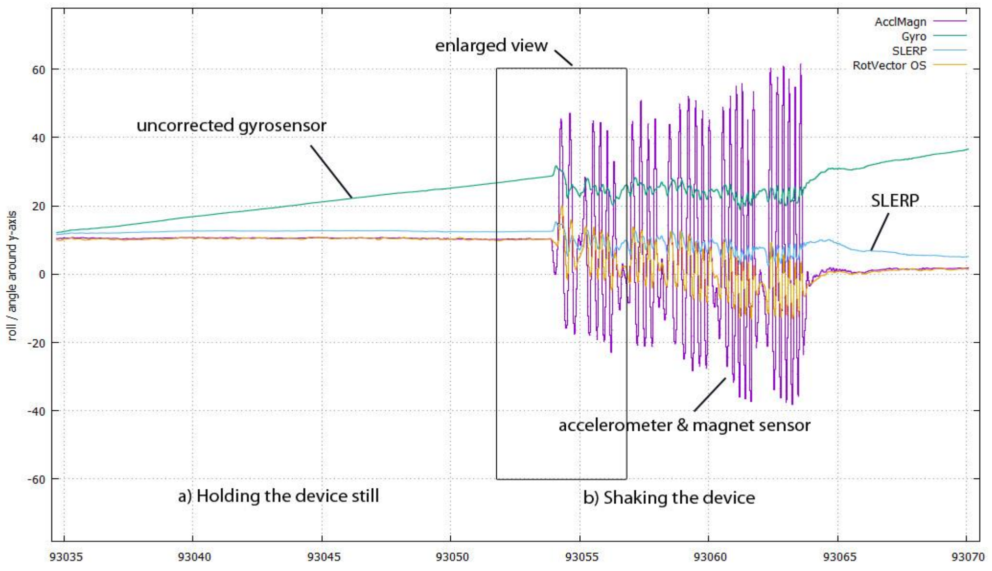


Figure 2 Exemplary plotted y-axis angle of different orientation estimations during holding still and shaking a mobile device

Figure 3 shows an englarged view of the plotted y-axis angle from the figure above.

It is visible that the SLERP interpolated orientation clearly follows the gyroscope-based curve but without the characteristic drift. Compared with the OS Rotation Vector our interpolated orientation is not influenced by the additional accelerative forces.

We've also tested our demo app on two older smartphones, a Galaxy Nexus and a Nexus S. Both smartphones are official Google development devices and they both perform differently. Our algorithm however performs on both of them better, more consistently and comparable than the stock Rotation Vector implementation.

The major advantage of our approach is the configurability of the sensor fusion process and the possibility to tweak the processing by means of individual needs. On the other hand it is hardly possible to tweak our algorithm to perform identically on every available device. Even though the SDK describes an uncalibrated gyroscope sensor, not every devices offers this sensor type. The default gyroscope sensor is being automatically corrected for drift. Under normal circumstances this produces quite good results. We tested our demo app on a consistently rotating turntable. First the orientation determination looked fine, but after some time the animated cubes stopped rotating. The constant rotating velocity had been detected misleadingly as a drift. After stopping the turntable our animated cubes started rotating in the opposite direction. Only one of our devices offers an uncalibrated gyroscope sensor and works correctly with our algorithm. On the other hand none of the Rotation Vector implementations of our devices performs correctly, as they all are apparently using the automatic drift compensation of Android.

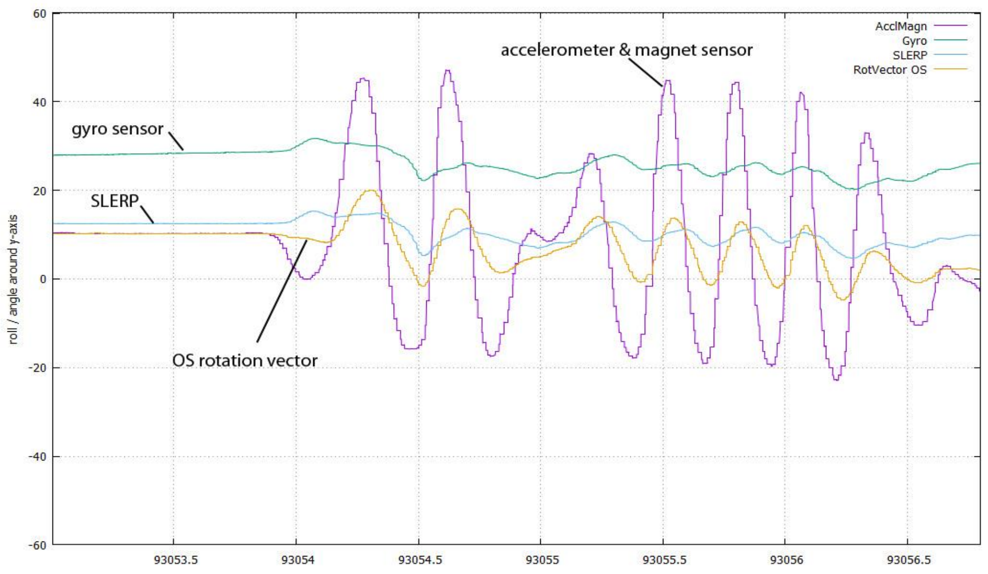


Figure 3 Enlarged view of y-axis angle plot

5 Related Work

Applications for smartphones based on motion detection with graphics need to be effective and efficient. These two factors can even depend on each other, e.g. if an algorithm is time-consuming. To improve efficiency, code can be optimized for sequential processing, or replaced by native code instead of Java, or code can be optimized for multicore processors or a graphical processing unit. E.g. the approach RAIOM (Mitaritonna & Abásolo 2013) provides a framework addressing different levels of AR applications, e.g. libraries for computer vision such as an embedded version of OPEN GL (Khronos 2015). This approach purely aims at computer vision and tracking only by camera based detection. Hence, such an approach could improve our graphics code but does not provide support for sensor fusion which is our primary focus.

Regarding effectiveness, we were looking for algorithms that fuse readings from different sensors to improve the orientation. Complementary filters, allegedly conceived around 1970 for aircraft control, have become more widespread recently since smartphones equipped with motion sensors and other affordable devices ranging from Kinect to robots are more widespread and offer easy to program SDKs for non-safety critical applications.

Theoretically, they are a kind of low pass filter for moving average. They are easier to understand and to program than Kalman filters (Kalman 1960) and computationally less costly. This lead us to adopting the complementary filter approach. A similar approach to ours can be found in the approach by Colton (Colton 2007). For a balancing application, he has implemented a sensor fusion using a gyrometer and one accelerometer for the x-axis based on the complementary filter. However, we also use y- and z-accelerometers and also include the compass.

Moreover, we choose quaternions to represent rotations, respectively orientations. In order to implement the complementary filter on two quaternions, we adopted the SLERP algorithm from Shoemake (Shoemake 1985). This is an algorithm for spherical linear interpolation for the purpose of animating 3D rotation using quaternions. Each interpolated point is a linear combination and lies on the same sphere. This algorithm matched exactly the requirement for a complementary filter on quaternions, or, to put it in another way, it turned out that the complementary filter on quaternions has an additional meaning or interpretation, which is the interpolation between two rotations.

As part of his master thesis (Lawitzki 2012), Lawitzki describes an Android implementation in Java which implements a fused orientation between gyroscope rotation and accelerometer/compass orientation using complementary filter. The approach is very similar to ours and differs in the following aspects. Firstly, it does not accumulate gyro readings before rotations are computed and it does not use multithreading. Thereby, it might lose some readings. Secondly, it implements the complementary filter on rotations matrices while we implement it using quaternions. Hence our filtering contains fewer lines of code but requires some translations or normalizations of quaternions.

It has also turned out, that very recently another similar algorithm was proposed by (Slupik et al. 2014), which they call lightweight quaternion filter, however not for Java. Also overall process of filtering is slightly different: Firstly, they estimate a new orientation based on the last orientation and the new gyroscope reading. Secondly, they compute an estimation based on accelerometers and magnetometers. But this is only used further if acceleration is not higher than a threshold. Thirdly, they use spherical linear interpolation to combine the first and second, the weighting again depends on the acceleration observed. They provide an initial comparison with extended Kalman filter and concluded that results were quite similar. In some cases, their approach performed even better, because they were able to eliminate certain accelerations through their threshold and weighting of pure acceleration. This is certainly an interesting option to include more dynamicity in the algorithm.

Another way to include more dynamicity could be an adaptive weighting between the two quaternions could be possibly realized with the help of a Kalman filter (Kalman 1960).

6 Conclusion & Outlook

In this paper, we have introduced an extension to the spherical linear interpolation algorithm. First measurements have demonstrated that our interpolated orientation clearly follows the gyroscope-based curve but without the characteristic drift. Compared with the operation system's Rotation Vector our interpolated orientation is not influenced by the additional accelerative forces.

Our multithreaded sensor fusion software architecture has arisen from the need for an efficient and configurable orientation determination algorithm with the future possibility of making comparisons between different implementations. As long as the extended configurability is not needed, it is easier to use the operating system's functionality instead. Our custom software architecture offers us a flexible foundation for future research in the field of sensor fusion.

The big variety of differently performing devices leads to an interesting future research area. It may be interesting to improve our algorithm in order to adapt automatically to different devices and their characteristics. The goal would be an identical performance on as many different devices as possible.

References

Chen, D., Schmidt, A. & Gellerson, H. (1999). An Architecture for Multi-Sensor Fusion in Mobile Environments. In (Eds.): *Proceedings of International Conference on Information Fusion*.

Colton, S. (2007). *The Balance Filter.* {https://b94be14129454da9cf7f056f5f8b89a9b17da0be .googledrive.com/host/0B0ZbiLZrqVa6Y2d3UjFVWDhNZms/filter.pdf}

Milette, G. & Stroud, A. (2012). *Professional Android Sensor Programming.* Wrox.

Mitaritonna, A. & Abásolo, M. J. (2013). *Hardware and software considerations for RAIOM Mobile Augmented Reality Framework.* {http://www.academia.edu/7948407/ Hardware_and_software_considerations_for_RAIOM_Mobile_Augmented_Reality_Framework}

Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. *Transaction of the ASME*, Journal of Basic Engineering, pages 35-45.

Khronos Group (2015). *Open GL ES.*{https://www.khronos.org/opengles/}

Lawitzki, P. (2012). *Master Thesis.* {http://www.thousand-thoughts.com/wp-content/uploads/MasterThesis_Lawitzki_2012.pdf}

Shoemake, K. (1985). Animating Rotation with Quaternion Curves. *Computer Graphics* Vol. 19, No 3.

Slupik, J. , Szczęsna, A. & Polański, A. (2014). Novel Lightweight Quaternion Filter for Determining Orientation Based on Indications of Gyroscope, Magnetometer and Accelerometer. In L.J. Chmielewski et al. (Eds.): *ICCVG 2014.* LNCS 8671, Springer International Publishing Switzerland, 586–593.