Android Sensor Fusion Tutorial

While working on my master thesis, I've made some experiences with sensors in Android devices and I thought I'd share them with other Android developers stumbling over my blog. In my work I was developing a head tracking component for a prototype system. Since it had to adapt audio output to the orientation of the users head, it required to respond quickly and be accurate at the same time.

I used my Samsung Galaxy S2 and decided to use its gyroscope in conjunction with the accelerometer and the magnetic field sensor in order to measure the user's head rotations both, quickly and accurately. To acheive this I implemented a complementary filter to get rid of the gyro drift and the signal noise of the accelerometer and magnetometer. The following tutorial describes in detail how it's done.

There are already several tutorials on how to get sensor data from the Android API, so I'll skip the details on android sensor basics and focus on the sensor fusion algorithm. The Android API Reference is also a very helpful entry point regarding the acquisition of sensor data. This tutorial is based on the Android API version 10 (platform 2.3.3), by the way.

This article is divided into two parts. The first part covers the theoretical background of a complementary filter for sensor signals as described by Shane Colton here. The second part describes the implementation in the Java programming laguage. Everybody who thinks the theory is boring and wants to start programing right away can skip directly to the second part. The first part is interesting for people who develop on other platforms than Android, iOS for example, and want to get better results out of the sensors of their devices.

Update (March 22, 2012):
I've created a small Android project which contains the whole runnable code from this tutorial. You can download it here:
SensorFusion1.zip

Update (April 4, 2012):
Added a small bugfix in the examples GUI code.

Update (July 9, 2012):
Added a bugfix regarding angle transitions between 179° <–> -179°. Special thanks to **J.W. Alexandar Qiu** who pointed it out and published the soultion!

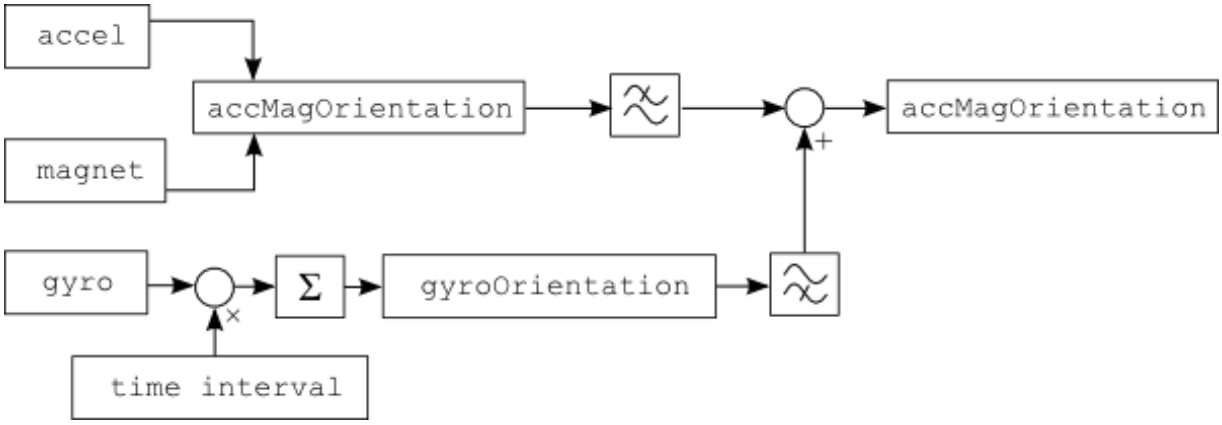
Update (September 25, 2012):
Published the code under the MIT-License (license note added in code), which allows you to do with it pretty much everything you want. No need to ask me first 😉

SENSOR FUSION VIA COMPLEMENTARY FILTER

Before we start programming, I want to explain briefly how our sensor fusion approach works. The common way to get the attitude of an Android device is to use the *SensorManager.getOrientation()* method to get the three orientation angles. These two angles are based on the accelerometer and magenotmeter output. In simple terms, the acceletometer provides the gravitiy vector (the vector pointing towards the centre of the earth) and the magnetometer works as a compass. The Information from both sensors suffice to calculate the device's orientation. However both sensor outputs are inacurate, expecially the output from the magnetic field sensor which includes a lot of noise.

The gyroscope in the device is far more accurate and has a very short response time. Its downside is the dreaded gyro drift. The gyro provides the angular rotation speeds for all three axes. To get the actual orientation those speed values need to be integrated over time. This is done by multiplying the angular speeds with the time interval between the last and the current sensor output. This yields a rotation increment. The sum of all rotation increments yields the absolut orientation of the device. During this process small errors are introduced in each iteration. These small errors add up over time resulting in a constant slow rotation of the calculated orientation, the gyro drift.

To avoid both, gyro drift and noisy orientation, the gyroscope output is applied only for orientation changes in short time intervals, while the magnetometer/acceletometer data is used as support information over long periods of time. This is equivalent to low-pass filtering of the accelerometer and magnetic field sensor signals and high-pass filtering of the gyroscope signals. The overall sensor fusion and filtering looks like this:



So what exactly does high-pass and low-pass filtering of the sensor data mean? The sensors provide their data at (more or less) regular time intervals. Their values can be shown as signals in a graph with the time as the x-axis, similar to an audio signal. The low-pass filtering of the noisy accelerometer/magnetometer signal (*accMagOrientation* in the above figure) are orientation angles averaged over time within a constant time window.

Later in the implementation, this is accomplished by slowly introducing new values from the accelerometer/magnetometer to the absolute orientation:

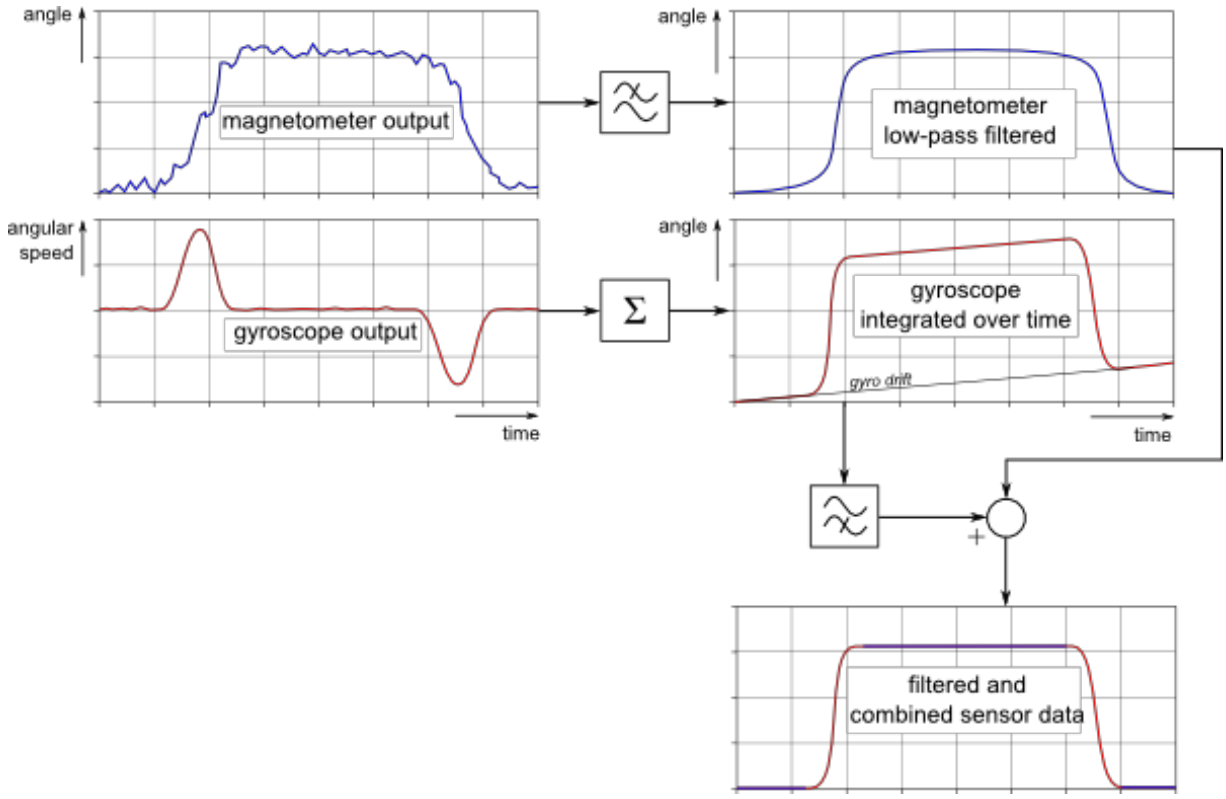
```
// low-pass filtering: every time a new sensor value is available
// it is weighted with a factor and added to the absolute orientation
accMagOrientation = (1 - factor) * accMagOrientation + factor * newAccMagValue;
```

The high-pass filtering of the integrated gyroscope data is done by replacing the filtered high-frequency component from *accMagOrientation* with the corresponding gyroscope orientation values:

```
fusedOrientation =
    (1 - factor) * newGyroValue;    // high-frequency component
    + factor * newAccMagValue;      // low-frequency component
```

In fact, this is already our finished comlementary filter.

Assuming that the device is turned 90° in one direction and after a short time turned back to its initial position, the intermediate signals in the filtering process would look something like this:



Notice the gyro drift in the integrated gyroscope signal. It results from the small irregularities in the original angular speed. Those little deviations add up during the integration and cause an additional undesireable slow rotation of the gyroscope based orientation.

127 RESPONSES TO *ANDROID SENSOR FUSION TUTORIAL*

Paul *says:*

2013-06-16 AT 5.40 PM

Hallo tuvbunn2,
was genau hast du denn schon versucht? Im Prinzip müsstest du die Rotationsmatrix in ein anderes Koordinatensystem umrechnen. Da ich mich schon länger nicht mehr mit dem Code auseinandergesetzt habe, müsste ich mich selber erst mal hinsetzen und darüber nachdenken (was ich leider zur Zeit nicht kann 😟).

tuvbunn2 *says:*

2013-06-19 AT 4.31 PM

Hi,

Ich habe eine Rotationsmatrix um die X achse gebaut und rotationMatrix damit in calculateAccMagOrientation() gedreht.

Leider funktioniert dann mein roll nicht mehr, bzw der Horizont korrigiert dann auf seltsame weise.

Das hier sind meine Anpassungen:

```
public void calculateAccMagOrientation()
{
if (SensorManager.getRotationMatrix(rotationMatrix, null, accel, magnet))
{
//rotation x axis 90 degrees
float[] xRotationMatrix = new float[9];
xRotationMatrix[0]=1f;
xRotationMatrix[1]=0f;
xRotationMatrix[2]=0f;
xRotationMatrix[2]=0f;
xRotationMatrix[0]=0f;
xRotationMatrix[5]=1f;
xRotationMatrix[6]=0f;
xRotationMatrix[7]=-1f;
xRotationMatrix[8]=0f;

rotationMatrix=matrixMultiplication(rotationMatrix, xRotationMatrix);

SensorManager.getOrientation(rotationMatrix, accMagOrientation);
}
}
```

Danke und Grüße,
tuvbunn2

Paul *says:*
2013-06-20 AT 10.34 PM

Hi tuvbunn2,
warum weist du xRotationMatrix[0] einmal 1f zu und überschreibst es weiter unten gleich wieder mit 0f? Ich glaube, dass du da einen copy-paste-Fehler in der Initialisierung deiner Rotationsmatrix hast, denn index 3 und 4 werden garnicht initialisiert, stattdessen 0 und 2 zweimal.

Außerdem reicht es nicht, wenn du allein den AccMag-Anteil rotierst. Den Gyro-Anteil (siehe gyroMatrix) musst du auch transformieren, sonst hast du später in der Sensor-Fusion inkonsistente Daten.

Grüße,
Paul

tuvbunn2 *says:*
2013-06-25 AT 11.49 AM

Oh vielen Dank,
Da ist wohl beim kopieren etwas durcheinander gekommen.
Durch die Rotationen der rotationMatrix um 90° sowie der Rotation der gyroMatix um 90° passt jetzt alles.

Vielen Dank.

Grüße,
tuvbunn2

PINGBACK: *GYROSCOPE ISSUES WITH DEVICE ORIENTATION | BLOGOSFERA*

Phillip *says:*
2013-07-26 AT 2.50 PM

Hi Paul,

This is so useful article!

But i have the same problem as tuvbunn2, i rotate the rotationMatrix by 90 before pasing to accMagOrientation which makes pitch work fine but the roll value deviate (no deviation for 45 degrees pitch though).

Where exactly in the alogrithm should i rotate gyro matrix?
I would be so grateful for helping me out with this

Paul *says:*
2013-07-26 AT 10.51 PM

Hi Phillip,
you hav to rotate both, the GyroMatrix and the AccMagMatrix (which is called rotationMatrix within the code) by 90° around the x-axis. you have to create a rotation matrix for that, just like tuvbunn2 did:

```
 1  0  0

0  0  1

0 -1  0
```

You can apply the rotation using the MatrixMultiplication-Method. The above rotation matrix has to be the second parameter (remember that matrix multiplications are not commutative). Just apply the rotation right after GyroMatrix/AccMagMatrix have been calculated respectively. For the AccMagMatrix that would be somewhere at the end of the method calculateAccMagOrientation and for the gyroMatrix somewhere at the end of the method gyroFunction.
Hope that helps.

Phillip *says:*
2013-07-27 AT 12.56 PM

Unfortunetely it doesnt work, after mulitplying gyroMatrix the values go crazy…
I am using your project and testing on few mobile devices:
float[] my90DegRotationMatrix = {1, 0, 0,
0, 0, 1,
0, -1, 0,
};

then in calculateAccMagOrientation:

```
if(SensorManager.getRotationMatrix(rotationMatrix, null, accel, magnet)) {
rotationMatrix = matrixMultiplication(rotationMatrix, my90DegRotationMatrix);
SensorManager.getOrientation(rotationMatrix, accMagOrientation);
}
```

and in gyroFunction at the very end:

```
// apply the new rotation interval on the gyroscope based rotation matrix
gyroMatrix = matrixMultiplication(gyroMatrix, deltaMatrix);

// rotate gyro matrix NEW
gyroMatrix = matrixMultiplication(gyroMatrix, my90DegRotationMatrix);

// get the gyroscope based orientation from the rotation matrix
SensorManager.getOrientation(gyroMatrix, gyroOrientation);
```

I tried rotating gyromatrx in few other places like at the end of calculateFusedOrientationTask etc. It seem like not matter where i do that it breaks everything.

Can you please help me, am i doing the gyro matrix multiplication wrong?

Paul *says:*
2013-07-30 AT 7.51 PM

Hi Phillip,
I'm sorry, but currently I don't have the project setup on my local machine so I cannot reproduce the behavior of your code. I was looking through the code and couldn't find anything, yet. I will report back, if I find something.

Jeff *says:*
2013-08-01 AT 4.40 PM

Hi Paul,
This article is amazing. Well done.

I am experiencing an issue I was hoping you can help with. In my own application and in your sample application, the Pitch value goes from 0 (device laying flat) to -90 (device completely vertical), then back down toward 0 (device flat but upside

down). This means the pitch value is -45 both when the phone is titled back (\) and when its tilted forward (/). I am using a Samsung Galaxy Nexus. Are these the values you would expect to see? How can I tell the difference between the two positions?

Thanks!

Paul *says:*
2013-08-02 AT 5.31 PM

Hi Jeff,
pitch, roll and azimuth always keep their relations to each other. That said, you can read the azimuth value in order to interpret which orientation, (/) or (\) is meant by a pitch value of -45°. If the reading direction (–>) is front, an azimuth of 0° and a pitch of -45° would be (\) (with the earpiece pointing down) while with an azimuth of 180° it would be (/). Of course if you flip the roll angle by 180° the whole situation is different. So I'm affraid you have to take into account all three angles.

oh *says:*
2013-08-12 AT 9.39 AM

It would be grand to apply this to android Dead Reckoning (continue to navigate (supplement) with weak gps or weak data (tower masts))

fariba *says:*
2013-08-17 AT 12.17 PM

Hi
The program is very good and very useful, thank you.
I've used your program in my project.Could you explain more about the TIME_CONSTANT and the FILTER_COEFFICIENT And how did you get them?

Paul *says:*
2013-08-18 AT 5.52 PM

Hi fariba,
thank you for your nice feedback. I think I already explained those two constants pretty much in detail further down in the comments. Try to look on page 2 of the comments. I believe there is a more lengthy explanation on this which I posted some time ago.

Pok *says:*
2013-08-19 AT 1.29 AM

Thank you for the tutorial Paul. Very useful. Is it possible to install it on Android 2.2 devices? Im getting a parsing error so just wondering it might be because of incompatibility (?)

Pok *says:*
2013-08-19 AT 1.45 AM

I think the reason is getrotationmatrixfromvector which has been added in API9.

Paul *says:*
2013-08-19 AT 5.29 PM

That could be. I didn't test the example with older API levels.

Rajesh *says:*
2013-09-02 AT 11.31 PM

Hi
Nice article ! I have a question !
How can I convert linear acceleration given by accelerometer sensor with the device frame of reference to the earth frame of reference ? I think it will need sensors (Acc, gyro, Mag) fusion to determine that.

From (Xa, Ya, Za) to (Xe, Ye, Ze) ?

please help

fariba *says:*
2013-09-09 AT 6.08 PM

Hey Paul
If I want to hold the phone in one of the following three conditions Should I change something in the code?
1 – when the x-axis parallel and opposite to the vector of gravity .
2 – when the y-axis parallel and opposite to the vector of gravity .
3 – when the z-axis parallel and opposite to the vector of gravity .
please explain what should I do For each case . Just have to change the rotation matrix? How?
please help me

ASHWINI *says:*
2013-10-06 AT 6.46 PM

Hello sir,

Thanks a lot for the tutorial,
actually am working on indoor tracking in gps denied areas using sensor fusion in android devices,
the source code you provided helps in gathering sensor values?
or fusing them as per sensor fusion so that they used for different purposes???

Android Example *says:*
2013-10-07 AT 12.35 PM

Very nice dude…

I have also found one good link here….

Accelerometer Basic Example – Detect Phone Shake Motion

Paul *says:*
2013-11-02 AT 6.58 PM

@ASHWINI: It’s basically sensor fusion, nothing more.

Aaron *says:*
2013-11-07 AT 2.41 PM

Hi Paul,
Good job.
I'm interested in Android PDR and can you send me the source code? I can't find the newest version of you code...

Paul *says:*
2013-11-11 AT 5.42 PM

Hi Aaron,

there is a link in the post after the fourth paragraph. The text starts wit "Update (March 22, 2012)". There you can download the code to this tutorial. Otherwise try this link.

Bartek *says:*
2013-11-19 AT 12.53 PM

Hello Paul,
Thanks for such a great tutorial!

I don't understand why are you using the magnetometer if it's so inaccurate and generate noise? Why not use accelerometer instead?

Even the accelerometer output look the same as magnetometer output.

Hao *says:*
2013-11-27 AT 1.35 AM

Dear Paul,

Can your also calculate the horizontal degree to the North?
Can it also be benefit from the filter your define?
Thanks.

Hao

Paul *says:*

2013-11-28 AT 12.29 AM

@Bartek:
The magnetometer is the only non-gyro reference for the azimuth angle we have available. You cannot determine the azimuth angle with the accelerometer, only pitch and yaw.
@Hao:
Could you be more specific? Do you mean the angle between the devices 'forward' vector and the noth direction? In 3D or projected on a horizontal plane?
The main problem would always be to determine north accurately enough. This could be problematic with the noisy magnetometer. You could point the device towards north using some other method (another accurate compass), save the azimuth angle (and thus 'calibrate' it, so to say) and calculate the difference between those two vectors. However, I have the feeling this would not be a satisfying solution to you.