Android Sensor Fusion Tutorial

IMPLEMENTATION

Now let's get started with the implementation in an actual Android application.

First we need to set up our Android app with the required members, get the *SensorManager* and initiaise our sensor listeners, for example, in the *onCreate* method:

```
1  public class SensorFusionActivity extends Activity implements SensorEventListener{
2      private SensorManager mSensorManager = null;
3
4      // angular speeds from gyro
5      private float[] gyro = new float[3];
6
7      // rotation matrix from gyro data
8      private float[] gyroMatrix = new float[9];
9
10     // orientation angles from gyro matrix
11     private float[] gyroOrientation = new float[3];
12
13     // magnetic field vector
14     private float[] magnet = new float[3];
15
16     // accelerometer vector
17     private float[] accel = new float[3];
18
19     // orientation angles from accel and magnet
20     private float[] accMagOrientation = new float[3];
21
22     // final orientation angles from sensor fusion
23     private float[] fusedOrientation = new float[3];
24
25     // accelerometer and magnetometer based rotation matrix
26     private float[] rotationMatrix = new float[9];
27     public void onCreate(Bundle savedInstanceState) {
28         super.onCreate(savedInstanceState);
29         setContentView(R.layout.main);
30
31         gyroOrientation[0] = 0.0f;
32         gyroOrientation[1] = 0.0f;
33         gyroOrientation[2] = 0.0f;
34
35         // initialise gyroMatrix with identity matrix
36         gyroMatrix[0] = 1.0f; gyroMatrix[1] = 0.0f; gyroMatrix[2] = 0.0f;
37         gyroMatrix[3] = 0.0f; gyroMatrix[4] = 1.0f; gyroMatrix[5] = 0.0f;
38         gyroMatrix[6] = 0.0f; gyroMatrix[7] = 0.0f; gyroMatrix[8] = 1.0f;
39
40         // get sensorManager and initialise sensor listeners
41         mSensorManager = (SensorManager) this.getSystemService(SENSOR_SERVICE);
42         initListeners();
43     }
44
45     // ...
46 }
```

Notice that the application implements the *SensorEventListener* interface. So we'll have to implement the two methods *onAccuracyChanged* and *onSensorChanged*. I'll leave *onAccuracyChanged* empty since it is not necessary for this tutorial. The more important function is *onSensorChanged*. It updates our sensor data continuously.

The initialisation of the sensor listeners happens in the *initListeners()* method:

```
public void initListeners(){
    mSensorManager.registerListener(this,
        mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
        SensorManager.SENSOR_DELAY_FASTEST);

    mSensorManager.registerListener(this,
        mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE),
        SensorManager.SENSOR_DELAY_FASTEST);

    mSensorManager.registerListener(this,
        mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD),
        SensorManager.SENSOR_DELAY_FASTEST);
}
```

After the listeners are initialised, the *onSensorChanged()* method is called automatically whenever new sensor data is available. The data is then copied or processed, respectively.

```
public void onSensorChanged(SensorEvent event) {
    switch(event.sensor.getType()) {
    case Sensor.TYPE_ACCELEROMETER:
        // copy new accelerometer data into accel array
        // then calculate new orientation
        System.arraycopy(event.values, 0, accel, 0, 3);
        calculateAccMagOrientation();
        break;

    case Sensor.TYPE_GYROSCOPE:
        // process gyro data
        gyroFunction(event);
        break;

    case Sensor.TYPE_MAGNETIC_FIELD:
        // copy new magnetometer data into magnet array
        System.arraycopy(event.values, 0, magnet, 0, 3);
        break;
    }
}
```

The Android API provides us with very handy functions to get the absolute orientation from the accelerometer and magnetometer. This is all we need to do to get the accelerometer/magnetometer based orientaion:

```
public void calculateAccMagOrientation() {
    if(SensorManager.getRotationMatrix(rotationMatrix, null, accel, magnet)) {
        SensorManager.getOrientation(rotationMatrix, accMagOrientation);
    }
}
```

As described above, the gyroscope data requires some additional processing. The Android reference page shows how to get a rotation vector from the gyroscope data (see Sensor.TYPE_GYROSCOPE). I have simply reused the proposed code and added some parameters to it so it looks like this:

```
public static final float EPSILON = 0.000000001f;

private void getRotationVectorFromGyro(float[] gyroValues,
                                       float[] deltaRotationVector,
                                       float timeFactor)
{
    float[] normValues = new float[3];

    // Calculate the angular speed of the sample
    float omegaMagnitude =
        (float)Math.sqrt(gyroValues[0] * gyroValues[0] +
        gyroValues[1] * gyroValues[1] +
        gyroValues[2] * gyroValues[2]);

    // Normalize the rotation vector if it's big enough to get the axis
    if(omegaMagnitude > EPSILON) {
        normValues[0] = gyroValues[0] / omegaMagnitude;
        normValues[1] = gyroValues[1] / omegaMagnitude;
        normValues[2] = gyroValues[2] / omegaMagnitude;
    }

    // Integrate around this axis with the angular speed by the timestep
    // in order to get a delta rotation from this sample over the timestep
    // We will convert this axis-angle representation of the delta rotation
    // into a quaternion before turning it into the rotation matrix.
    float thetaOverTwo = omegaMagnitude * timeFactor;
    float sinThetaOverTwo = (float)Math.sin(thetaOverTwo);
    float cosThetaOverTwo = (float)Math.cos(thetaOverTwo);
    deltaRotationVector[0] = sinThetaOverTwo * normValues[0];
    deltaRotationVector[1] = sinThetaOverTwo * normValues[1];
    deltaRotationVector[2] = sinThetaOverTwo * normValues[2];
    deltaRotationVector[3] = cosThetaOverTwo;
}
```

The above function creates a rotation vector which is similar to a quaternion. It expresses the rotation interval of the device between the last and the current gyroscope measurement. The rotation speed is multiplied with the time interval — here it's the parameter *timeFactor* — which passed since the last measurement. this function is then called in the actual *gyroFunction()* for gyro sensor data processing. This is where the gyroscope rotation intervals are added to the absolute gyro based orientation. But since we have rotation matrices instead of angles this can't be done by simply adding the rotation intervals. We need to apply the rotation intervals by mytrix multiplication:

```
private static final float NS2S = 1.0f / 1000000000.0f;
private float timestamp;
private boolean initState = true;

public void gyroFunction(SensorEvent event) {
    // don't start until first accelerometer/magnetometer orientation has been acquired
    if (accMagOrientation == null)
        return;

    // initialisation of the gyroscope based rotation matrix
    if(initState) {
        float[] initMatrix = new float[9];
        initMatrix = getRotationMatrixFromOrientation(accMagOrientation);
        float[] test = new float[3];
        SensorManager.getOrientation(initMatrix, test);
        gyroMatrix = matrixMultiplication(gyroMatrix, initMatrix);
        initState = false;
    }

    // copy the new gyro values into the gyro array
    // convert the raw gyro data into a rotation vector
    float[] deltaVector = new float[4];
    if(timestamp != 0) {
        final float dT = (event.timestamp - timestamp) * NS2S;
    System.arraycopy(event.values, 0, gyro, 0, 3);
    getRotationVectorFromGyro(gyro, deltaVector, dT / 2.0f);
```

```
    }

    // measurement done, save current time for next interval
    timestamp = event.timestamp;

    // convert rotation vector into rotation matrix
    float[] deltaMatrix = new float[9];
    SensorManager.getRotationMatrixFromVector(deltaMatrix, deltaVector);

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

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

The gyroscope data is not processed until orientation angles from the accelerometer and magnetometer is available (in the member variable *accMagOrientation*). This data is required as the initial orientation for the gyroscope data. Otherwise, our orientation matrix will contain undefined values. The device's current orientation and the calcuated gyro rotation vector are transformed into a rotation matrix.

The *gyroMatrix* is the total orientation calculated from all hitherto processed gyroscope measurements. The *deltaMatrix* holds the last rotation interval which needs to be applied to the *gyroMatrix* in the next step. This is done by multiplying *gyroMatrix* with *deltaMatrix*. This is equivalent to the Rotation of *gyroMatrix* about *deltaMatrix*. The *matrixMultiplication* method is described further below. Do not swap the two parameters of the matrix multiplication, since matrix multiplications are not commutative.

The rotation vector can be converted into a matrix by calling the conversion function *getRotationMatrixFromVector* from the *SensoManager*. In order to convert orientation angles into a rotation matrix, I've written my own conversion function:

```
private float[] getRotationMatrixFromOrientation(float[] o) {
    float[] xM = new float[9];
    float[] yM = new float[9];
    float[] zM = new float[9];

    float sinX = (float)Math.sin(o[1]);
    float cosX = (float)Math.cos(o[1]);
    float sinY = (float)Math.sin(o[2]);
    float cosY = (float)Math.cos(o[2]);
    float sinZ = (float)Math.sin(o[0]);
    float cosZ = (float)Math.cos(o[0]);

    // rotation about x-axis (pitch)
    xM[0] = 1.0f; xM[1] = 0.0f; xM[2] = 0.0f;
    xM[3] = 0.0f; xM[4] = cosX; xM[5] = sinX;
    xM[6] = 0.0f; xM[7] = -sinX; xM[8] = cosX;

    // rotation about y-axis (roll)
    yM[0] = cosY; yM[1] = 0.0f; yM[2] = sinY;
    yM[3] = 0.0f; yM[4] = 1.0f; yM[5] = 0.0f;
    yM[6] = -sinY; yM[7] = 0.0f; yM[8] = cosY;

    // rotation about z-axis (azimuth)
    zM[0] = cosZ; zM[1] = sinZ; zM[2] = 0.0f;
    zM[3] = -sinZ; zM[4] = cosZ; zM[5] = 0.0f;
    zM[6] = 0.0f; zM[7] = 0.0f; zM[8] = 1.0f;

    // rotation order is y, x, z (roll, pitch, azimuth)
```

```
29      float[] resultMatrix = matrixMultiplication(xM, yM);
30      resultMatrix = matrixMultiplication(zM, resultMatrix);
31      return resultMatrix;
32  }
```

I have to admit, this function is not optimal and can be improved in terms of performance, but for this tutorial it will do the trick. It basically creates a rotation matrix for every axis and multiplies the matrices in the correct order (y, x, z in our case).

This is the function for the matrix multiplication:

```
1   private float[] matrixMultiplication(float[] A, float[] B) {
2       float[] result = new float[9];
3
4       result[0] = A[0] * B[0] + A[1] * B[3] + A[2] * B[6];
5       result[1] = A[0] * B[1] + A[1] * B[4] + A[2] * B[7];
6       result[2] = A[0] * B[2] + A[1] * B[5] + A[2] * B[8];
7
8       result[3] = A[3] * B[0] + A[4] * B[3] + A[5] * B[6];
9       result[4] = A[3] * B[1] + A[4] * B[4] + A[5] * B[7];
10      result[5] = A[3] * B[2] + A[4] * B[5] + A[5] * B[8];
11
12      result[6] = A[6] * B[0] + A[7] * B[3] + A[8] * B[6];
13      result[7] = A[6] * B[1] + A[7] * B[4] + A[8] * B[7];
14      result[8] = A[6] * B[2] + A[7] * B[5] + A[8] * B[8];
15
16      return result;
17  }
```


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.