**[Android: Algorithms for SensorManager.getRotationMatrix and SensorManager.getOrientation()](https://stackoverflow.com/questions/32372847/android-algorithms-for-sensormanager-getrotationmatrix-and-sensormanager-getori)**

To get orientation as a from of euler angles (e.g., pitch, roll, azimuth) in Android, it is required to execute followings:

1. SensorManager.getRotationMatrix(float[] R, float[] I, float[] gravity, float[] geomagnetic);
2. SensorManager.getOrientation(float[] R, float[] orientation);

In the first one, I realize that it uses a kind of TRIAD algorithms; Rotation matrix (R[]) is composed of gravity, geomagnetic X gravity, gravity X (geomagnetic X gravity) --- X is cross product.   
See codes below:

float Ax = gravity[0];

float Ay = gravity[1];

float Az = gravity[2];

final float Ex = geomagnetic[0];

final float Ey = geomagnetic[1];

final float Ez = geomagnetic[2];

float Hx = Ey\*Az - Ez\*Ay;

float Hy = Ez\*Ax - Ex\*Az;

float Hz = Ex\*Ay - Ey\*Ax;

final float normH = (float)Math.sqrt(Hx\*Hx + Hy\*Hy + Hz\*Hz);

if (normH < 0.1f) {

// device is close to free fall (or in space?), or close to

// magnetic north pole. Typical values are > 100.

return false;

}

final float invH = 1.0f / normH;

Hx \*= invH;

Hy \*= invH;

Hz \*= invH;

final float invA = 1.0f / (float)Math.sqrt(Ax\*Ax + Ay\*Ay + Az\*Az);

Ax \*= invA;

Ay \*= invA;

Az \*= invA;

final float Mx = Ay\*Hz - Az\*Hy;

final float My = Az\*Hx - Ax\*Hz;

final float Mz = Ax\*Hy - Ay\*Hx;

if (R != null) {

if (R.length == 9) {

R[0] = Hx; R[1] = Hy; R[2] = Hz;

R[3] = Mx; R[4] = My; R[5] = Mz;

R[6] = Ax; R[7] = Ay; R[8] = Az;

} else if (R.length == 16) {

R[0] = Hx; R[1] = Hy; R[2] = Hz; R[3] = 0;

R[4] = Mx; R[5] = My; R[6] = Mz; R[7] = 0;

R[8] = Ax; R[9] = Ay; R[10] = Az; R[11] = 0;

R[12] = 0; R[13] = 0; R[14] = 0; R[15] = 1;

}

}

However, I cannot understand SensorManager.getOrientation().

azimuth = (float)Math.atan2(R[1], R[4]);

pitch = (float)Math.asin(-R[7]);

roll = (float)Math.atan2(-R[6], R[8]);

What is the exact algorithms to get euler angles?

Let me try to explain: getRotationMatrix compose rotation matrix on the basis of gravity and megnetic vector.

Our main goal here is to construct [NED frame](https://en.wikipedia.org/wiki/North_east_down)

We assume that gravity points toward the center of the Earth and magnet to the north Pole. But in real cases these vectors are non-perpendicular, that's why we firstly calculate vector H that is orthogonal to E and A and belong to tangential plane. H is a cross-product (E x A) and is orthogonal to E and A.

float Hx = Ey\*Az - Ez\*Ay;

float Hy = Ez\*Ax - Ex\*Az;

float Hz = Ex\*Ay - Ey\*Ax;

final float normH = (float)Math.sqrt(Hx\*Hx + Hy\*Hy + Hz\*Hz);

normalize acceleration and H vector (because these vectors will compose a basis of ENU coordinate system)

final float invH = 1.0f / normH;

Hx \*= invH;

Hy \*= invH;

Hz \*= invH;

final float invA = 1.0f / (float)Math.sqrt(Ax\*Ax + Ay\*Ay + Az\*Az);

Ax \*= invA;

Ay \*= invA;

Az \*= invA;

Find last basis vector (M) as cross-product of H and A:

double Mx = Ay \* Hz - Az \* Hy;

double My = Az \* Hx - Ax \* Hz;

double Mz = Ax \* Hy - Ay \* Hx;

Coordinates of the arbitrary vector (a) in body frame expresses through NED coordinates as a = Ra' R - Transformation matrix matrix, whose columns are the coordinates of the new basis vectors in the old basis

But coordinates in NED frame are calculated as a' = T^(-1) \* a. For orthogonal transformation matrix inverse is equal to transposed matrix. Thus we have:

R[0] = Hx; R[1] = Hy; R[2] = Hz;

R[3] = Mx; R[4] = My; R[5] = Mz;

R[6] = Ax; R[7] = Ay; R[8] = Az;

Once we have rotation matrix we can convert it to Euler angles representation. Formulas of convertion depend from [convention](https://en.wikipedia.org/wiki/Euler_angles#Conventions) that you use. Your formulas

azimuth = (float)Math.atan2(R[1], R[4]);

pitch = (float)Math.asin(-R[7]);

roll = (float)Math.atan2(-R[6], R[8]);

are correct for Tiat Bryan angles with convention Y-X-Z. In order to better understand conversion from rotation matrix to Euler angles I would suggest to study an article of Gregory [G. Slabaugh - "Computing Euler angles from a rotation matrix"](http://staff.city.ac.uk/%7Esbbh653/publications/euler.pdf)

I am going to give a geometric interpretation of getOrientation and explain how you can calculate all kind of rotations using only RotationMatrix. If you understand what getOrientation does then there is no need to use it and if you don't then using it can give you all kind of troubles.

Specifically, it will give answer to many questions concerning orientation posted at Stack Exchange such as

* Writing a compass for tablet in Portrait mode. Why does after calling getRotation one needs to add 90 degree to get the correct answer.
* Why does RemapCoordinateSystem has to be called before calling getOrientation to obtain the direction of the back camera (negative of the device z-axis).
* Why after calling RemapCoordinateSystem and then getOrientation to correctly obtain the direction of the device z-axis, when the device is flat the result does not make sense anymore.
* How to calculate the rotation about any device axis independent of the device position i.e. flat or not flat.

First I need to explain RotationMatrix in more detail and what can it does before explaining getOrientation

There is 2 coordinates systems to be considered here.  
One is the World coordinates system with the x-axis pointing East, y-axis pointing North and the z-axis pointing to the sky.  
The other is the device coordinates system with the x-axis the shorter side on a phone (longer side on a tablet), y-axis the longer side on a phone (shorter side on a tablet) and z-axis is the vector orthogonal to the screen pointing out at you.

Mathematically speaking our object is the 3 dimensional real vector space and we are using the following bases for this vector space.

World basis W = {E, N, SKY} where  
E is a unit vector lying in the East direction  
N is a unit vector lying in the North direction  
SKY is a unit vector lying in the Sky direction

Device basis D = {X, Y, Z} where  
X is a unit vector lying in the phone short side direction (long side for tablet)  
Y is a unit vector lying in the phone long side direction (short side for tablet)  
Z is a unit vector lying in the direction orthogonal to the screen pointing out at you

For those who forgot what a basis is, it means every vector **v** in 3 space can be written as

In World basis  
**v** = a\_1 **E** + a\_2 **N** + a\_3 **SKY** where a\_1, a\_2, a\_3 are real number  
And usually written as (a\_1, a\_2, a\_3)\_W or just (a\_1, a\_2, a\_3). This 3 tuples is called the coordinate of **v** with respect to World basis or just coordinate of **v** with the basis implicit understood to be the World basis

In Device basis  
**v** = b\_1 **X** + b\_2 **Y** + b\_3 **Z** where b\_1, b\_2, b\_3 are real number  
And usually written as (b\_1, b\_2, b\_3)\_W or just (b\_1, b\_2, b\_3). This 3 tuples is called the coordinate of **v** with respect to Device basis or just coordinate of **v** with the basis implicitly understood to be the Device basis. Values returned by sensors are in Device basis, for example accelerometer returned as values[0], values[1] and values[2] written in Device basis would be  
**acc** = values[0] **X** + values[1] **Y** + values[2] **Z**

In particular  
**X** = 1 **X** + 0 **Y** + 0 **Z**  
**Y** = 0 **X** + 1 **Y** + 0 **Z**  
**Z** = 0 **X** + 0 **Y** + 1 **Z**

that is the coordinate of **X**, **Y** and **Z** with respect to the device basis is (1, 0, 0), (0, 1, 0) and (0, 0, 1) respectively. You will see how these vectors will be used to explain the calculation in getOrientation later.

Notice that the World basis is fixed, but the Device basis changes as the position of the phone changes. That is the unit vectors **X, Y, Z** change as the device position changes. They are defined in a same way but they are different vectors when position changes. You still write them as **X, Y, Z** but they are different **X, Y, Z**.  
Thus if a phone stands still the only force acting on the phone is gravity and hence the accelerometer vector theoretically is a vector lying on the World Sky axis, that is in World basis the coordinate is (0, 0, g). In device basis it is (a\_1, a\_2, a\_3) for some a\_1, a\_2 and a\_3. If the device stands still at a different orientation the accelerometer is still the same that is (0, 0, g) in World basis but now it is (b\_1, b\_2, b\_3) in Device basis where at least one the a's is different from the b's.

Among the parameters for getRotationMatrix are the gravity and geomagnetic. gravity is assumed to be real gravity that is it coordinate is (0, 0, k) in World basis and geomagnetic is assume to lies on the World N-Sky plane that is it coordinate is (0, a, b) in World basis.

With these assumption let see how the Rotation Matrix is calculated in getRotationMatrix. What it tries in this method is to obtain a matrix **M** such that given any vector with coordinate (a\_1, a\_2, a\_3) in Device basis the product **M** (a\_1, a\_2, a\_3)\_T (transpose) gives the coordinate (b\_1, b\_2, b\_3) in World basis. Mathematically speaking the getRotationMatrix calculate the change of basis matrix.

The passed in parameters for gravity **g** should be the values obtained from onSensorChanged for TYPE\_GRAVITY or low pass filter of TYPE\_ACCELEROMETER and for geomagnetic **m** should be the values from TYPE\_MAGNETIC\_FIELD

**g** = a\_1 **X** + a\_2 **Y** + a\_3 **Z**  
**m** = b\_1 **X** + b\_2 **Y** + b\_3 **Z**

Without loss of generality assume **g** and **m** are already normalize, that is the norm of **g** and **m** are equal to 1. What we want to do now is to write the World basis {E, N, SKY} in Device basis.

Since **g** is assume to be gravity, that is (0, 0, 1) in World basis this is exactly the vector **SKY**.

**SKY** = **g** = a\_1 **X** + a\_2 **Y** + a\_3 **Z**  
That is the coordinate of **SKY** in Device basis is (a\_1, a\_2, a\_3)

Now since **m** is assumed to lies in the World N-SKY plane the cross product of **m** and **g** is a unit vector orthogonal to the World N-SKY plane and point to the right and thus this is E.

**E** = **m** x **g** = b\_1 **X** + b\_2 **Y** + b\_3 **Z**

That is the coordinate of **E** with respect to the Device basis is (b\_1, b\_2, b\_3) the b's are the values resulting from the cross product of **m** and **g**

Finally the cross product of **SKY** and **E** is a vector orthogonal to the E-SKY plane and is **N**

**N** = **SKY** x **E** = c\_1 **X** + c\_2 **Y** + c\_3 **Z**

Putting together we have

**E** = b\_1 **X** + b\_2 **Y** + b\_3 **Z**  
**N** = c\_1 **X** + c\_2 **Y** + c\_3 **Z**  
**SKY** = a\_1 **X** + a\_2 **Y** + a\_3 **Z**

Thus given any coordinate in World basis we can find the coordinate in Device basis. For example (1, 2, 3) that is a vector **v**

**v** = **E** + **2** N + **3** SKY

written in Device coordinate would be by substituting and multiply out

**v** = b\_1 **X** + b\_2 **Y** + b\_3 **Z** + 2 (c\_1 **X** + c\_2 **Y** + c\_3 **Z**) + 3 (a\_1 **X** + a\_2 **Y** + a\_3 **Z**)  
**v** = (b\_1 + 2 c\_1 + 3 a\_1) **X** + (b\_2 + 2 c\_2 + 3 a\_2) **Y** + (b\_3 + 2 c\_3 + 3 a\_3) **Z**

But we are really interested in the reverse which is given any coordinate in Device basis find the coordinate in the World basis. Well for those who do not remember linear algebra, in the above we have a 3 equations in 3 unknown thus we should be able to solve this equation for **X**, **Y** and **Z** in term of **E**, **N** and **SKY**.

Mathematically speaking the matrix obtain by putting the a,b and c above in colunms is the change of basis matrix from the World basis to the Device basis and thus we need to find the inverse of it. But this matrix is an orthonormal matrix and thus its inverse is just its transpose.

Written in code

a[0] a[1] a[2]

a[3] a[4] a[5]

a[6] a[7] a[8]

given any coordinate (a\_1, a\_2, a\_3) in the Device basis we can find the coordinate (b\_1, b\_2, b\_3) in the World coordinate by taking the product of the matrix above and the transpose of (a\_1, a\_2, a\_3).

In particular

a[0] a[1] a[2] 1 a[0]

a[3] a[4] a[5] x 0 = a[3]

a[6] a[7] a[8] 0 a[6]

Thus the coordinates of **X** in World basis is (a[0], a[3], a[6])

a[0] a[1] a[2] 0 a[1]

a[3] a[4] a[5] x 1 = a[4]

a[6] a[7] a[8] 0 a[7]

Thus the coordinates of **Y** in World basis is (a[1], a[4], a[7])

a[0] a[1] a[2] 0 a[2]

a[3] a[4] a[5] x 0 = a[5]

a[6] a[7] a[8] 1 a[8]

Thus the coordinates of **Z** in World basis is (a[2], a[5], a[8])

Now let see what getOrientation does.

The code for it is

azimuth = (float)Math.atan2(R[1], R[4]);

pitch = (float)Math.asin(-R[7]);

roll = (float)Math.atan2(-R[6], R[8]);

And the document say

values[0]: azimuth, rotation around the -Z axis, i.e. the opposite direction of Z axis.  
values[1]: pitch, rotation around the -X axis, i.e the opposite direction of X axis.  
values[2]: roll, rotation around the Y axis.

Take the above document literally without considering the position of the device is wrong. The document is true for the azimuth only if the device is flat.

Before we continue let me note that all calculations have to be done with coordinates relative to the same basis. For example if you want find the angle between 2 vectors, the coordinates of these 2 vectors had to be with respect to the same basis.

Now when you calculate rotation, you kind of implicitly understand that the rotation is a deviation from a fixed position. That is if the answer is 90 degrees then it is 90 degree from what? In the case of the azimuth it is 90 degree from the magnetic north, that is you calculate the deviation from the **N** vector. If you cannot spell out this implicit fixed position, you are in for a lot of troubles. For example what is the fixed vector for the pitch? roll? device orientation (portrait-landscape)?

Let take a look at each calculation and see what it really calculates.

For the azimuth the official calculation is

azimuth = (float)Math.atan2(R[1], R[4]);

Let go back to the way **Y** is written in Device coordinate. As mention earlier the coordinate of **Y** is (0, 1, 0) in the Device basis and the coordinate of **Y** in the World basis is (a[1], a[4], a[7])

The coordinate in World basis of the projection of **Y** into the World XY plane is (a[1], a[4]). The angle between this projection vector and the **N** vector is (draw the picture of the projection vector and N if you do not get it)

Math.atan2(R[1], R[4]);

which is precisely the calculation for the azimuth.

Thus getOrientation calculate the angle between the projection of the device y-axis into the World XY-plane and the World north axis. Hence if the device is held vertically, this calculation does not make sense since the y-coordinate of the projection is always 0. Geometrically it does not make sense to calculate the direction due north if you are pointing at the sky. Also, in this case rotation around the -Z axis would mean rotation away from Portrait so the document is not correct.

For Pitch the official calculation is

pitch = (float)Math.asin(-R[7]);

The coordinate in the World basis of the projection of the **Y** vector into the World N-SKY plane is (a[4], a[7]) and the angle between this projection vector and the projection of **Y** vector into the world E-N plane which is the same as the angle between the projection of the **Z** vector into the Y-Z plane and the gravity vector is (again draw picture for yourself)

Math.asin(-R[7])

Thus the pitch is the angle between the projection of the device y-axis into the World N-SKY plane and the projection of **Y** into the world E-N plane

Similarly the Roll is the angle between the projection of the device x-axis into the World X-SKY plane and the projection of **X** into the world E-N plane.

For example I pointed out earlier.

* Writing a compass for tablet in Portrait mode. Why does after calling getRotation one needs to add 90 degree to get the correct answer.

In this case one wants to calculate the angle between the projection of the x-axis into the World XY plane and thus one need to add 90 degrees as the angle between the x and y axes is always 90 degrees. One can instead get the coordinate in World basis of the X vector which is (a[0], a[3], a[6]) and then project into the World XY plane to get (a[0], a[3]) and do the calculation which is Math.atan2(R[0], R[3]).

* Why does RemapCoordinateSystem has to be called before calling getOrientation to obtain the direction of the back camera (negative of the device z-axis).

In this case you want to calculate the direction of the z axis and not the y axis which the getOrientation calculates. Thus you have to call remapCoordinateSystem which maps the Z axis into the Y axis. Geometrically, what you do is to rotate the z-axis to the y-axix and now y-axis becomes -z-axis. Thus you need to negate the y-column in the Rotation Matrix to get back to the original define coordinates system. You can just get the coordinate of in World basis of the **-Z** which is (-a[2], -a[5], -a[8]) and then project into the World XY plane to get (-a[2], -a[5]) and do the calculation.

**Note:** I first work with the Sensor's classes in 2011 which after knowing what the getRotationMatrix and getOrientation do, I created a library to do the calculation using only Rotation Matrix. I did not think remapCoordinateSystem and getOrientation are needed, but I did not realize they are bad to use until I write this answer, since it will give the correct azymuth but then give the wrong value for pitch. What remapCoordinateSystem does is to make the calculation of the azymuth in getOrientation correct according to the direction you want to calculate. But then the pitch value is incorrect you have to add -90 degrees to it to get the right value for the case of back camera direction and whatever it maybe for other cases.

If I were to write this class I would create

float getPitch(float[] rotMatrix)

float getRoll(float[] rotMatrix)

float getOrientation(float[] rotMatrix, int axis\_you\_want\_to\_calculate)

Then there will be no confusion and all the answers would be correct.

* Why after calling RemapCoordinateSystem and then getOrientation to correctly obtain the direction of the device z-axis, when the device is flat the result does not make sense anymore.

You are trying to calculate the direction of the -z axis due north and now it is pointing to the sky so the calculation does not make sense anymore.

* How to calculate the rotation about any device axis independent of the device position i.e. flat or not flat.

Tomorrow I will post an answer for the case of rotation about the z axis at [Get Euler Yaw angle of an Android Device](https://stackoverflow.com/questions/35303111/get-euler-yaw-angle-of-an-android-device).

Note: If you just want to write a compass app and the device is always flat then TYPE\_ORIENTATION is still good.

Note 2: I am terrible at drawing and thus cannot post any picture to illustrate. I can not even draw a straight line using Gimp (I followed instruction to hold down the shift key but my line was ragged). If someone is good at drawing and willing to help just drop a comment and I will give instruction of what the picture illustrations I had in mind and where to put them.