EXHIBIT 1

CYWEE GROUP LTD,

vs.

LG ELECTRONICS, INC.,
LG ELECTRONICS U.S.A., INC.,
AND LG ELECTRONICS MOBILECOMM U.S.A., INC.,

UNITED STATES DISTRICT COURT
FOR THE SOUTHERN DISTRICT OF CALIFORNIA

EXEMPLARY CLAIM CHART

U.S. PATENT NO. 8,441,438 – LG V20
Infringement Contentions

These contentions are disclosed to only provide notice of Plaintiff's theories of infringement. These contentions do not constitute proof nor do they marshal Plaintiff's evidence of infringement to be presented during trial.

Claim 1

Claim 1, with claim constructions, is recited below (text in brackets [] reflects the Court's claim construction or the parties' agreed claim construction in *CyWee Group, Ltd. v. Apple Inc.*, No. 3:13-cv-01853-HSG or in *CyWee Group, Ltd. v. Samsung Elecs. Co., Ltd.*, No. 2:17-cv-00140-WCB). Construed terms and constructions are underlined.

1. A three-dimensional (3D) pointing device [*Samsung* Court's construction: no construction necessary] subject to movements and rotations in dynamic environments, comprising:

a housing associated with said movements and rotations of the 3D pointing device in a spatial pointer reference frame [*Samsung* Court's construction: frame of reference associated with the 3D pointing device, which always has its origin at the same point in the device and in which the axes are always fixed with respect to the device];

a printed circuit board (PCB) enclosed by the housing;

a six-axis motion sensor module [*Samsung* Court's construction: no construction necessary] attached to the PCB, comprising a rotation sensor for detecting and generating a first signal set [*Samsung* Court's construction: no construction necessary] comprising angular velocities ω_x, ω_y, ω_z associated with said movements and rotations of the 3D pointing device in the spatial pointer reference frame, an accelerometer for detecting and generating a second signal set [*Samsung* Court's construction: no construction necessary] comprising axial accelerations A_x, A_y, A_z associated with said movements and rotations of the 3D pointing device in the spatial pointer reference frame; and

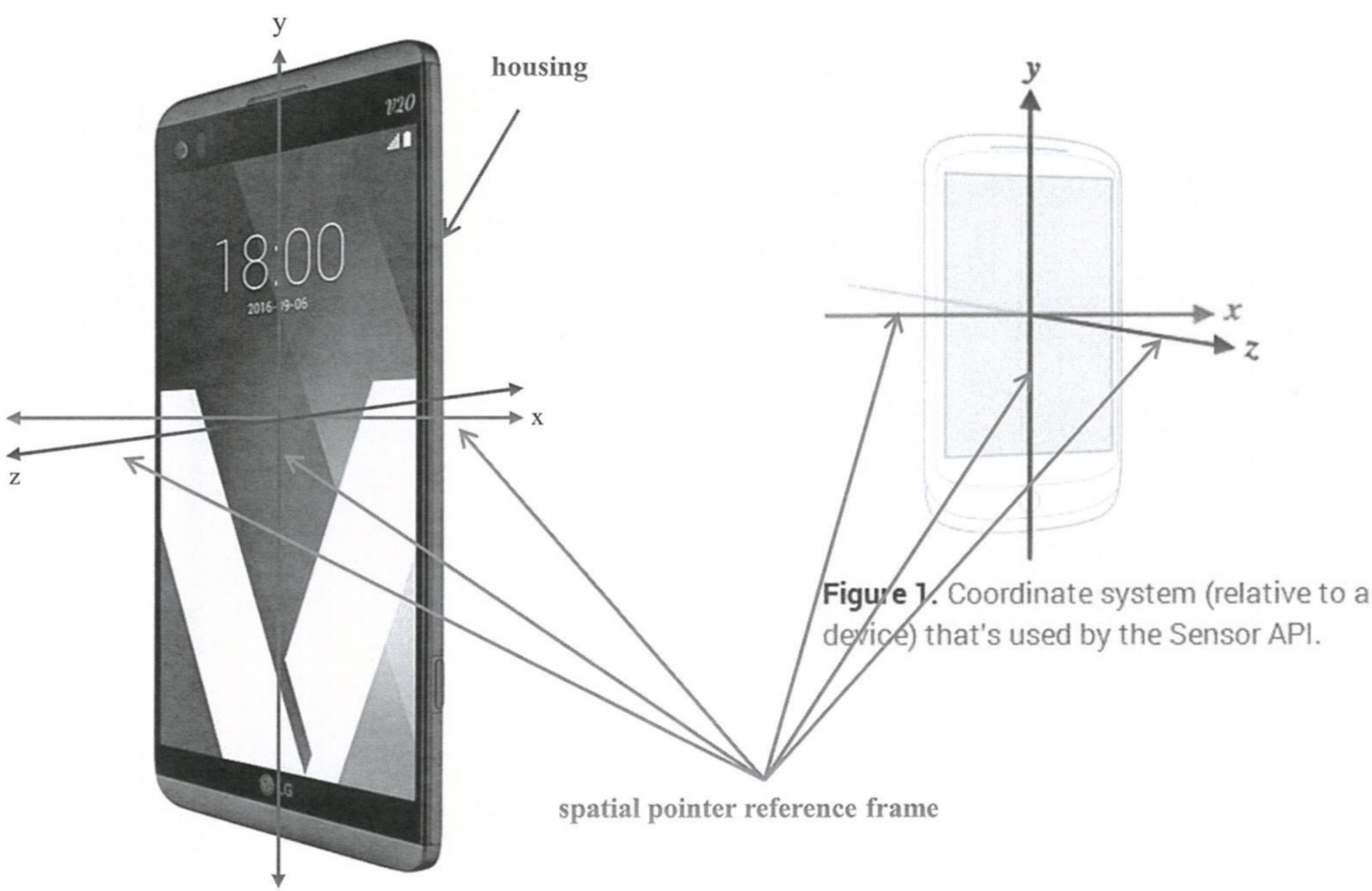
a processing and transmitting module, comprising a data transmitting unit electrically connected to the six-axis motion sensor module for transmitting said first and second signal sets thereof and a computing processor for receiving and calculating said first and second signal sets from the data transmitting unit [*Apple* Court's construction: no construction necessary], communicating with the six-axis motion sensor module to calculate a resulting deviation comprising resultant angles in said spatial pointer reference frame [*Samsung* Court's construction: no construction necessary] by utilizing a comparison to compare the first signal set with the second signal set [*Apple* Court's construction: using the calculation of actual deviation angles to compare the first signal set with the second signal set] whereby said resultant angles in the spatial pointer reference frame of the resulting deviation of the six-axis motion sensor module of the 3D pointing device are obtained under said dynamic environments, wherein the comparison utilized by the processing and transmitting module further comprises an update program to obtain an updated state based on a previous state associated with said first signal set and a measured state associated with said second signal set; wherein the measured state includes a measurement of said second signal set and a predicted measurement obtained based on the first signal set without using any derivatives of the first signal set [*Samsung* Court's construction: the measured state includes a measurement of axial accelerations and predicted axial accelerations calculated using the angular velocities without computing derivatives of said angular velocities (i.e. angular accelerations)].

U.S. Patent No. 8,441,438 – LG V20

Claim 1
A three-dimensional (3D) pointing device [*Samsung* Court's construction: no construction necessary] subject to movements and rotations in dynamic environments, comprising:

LG V20

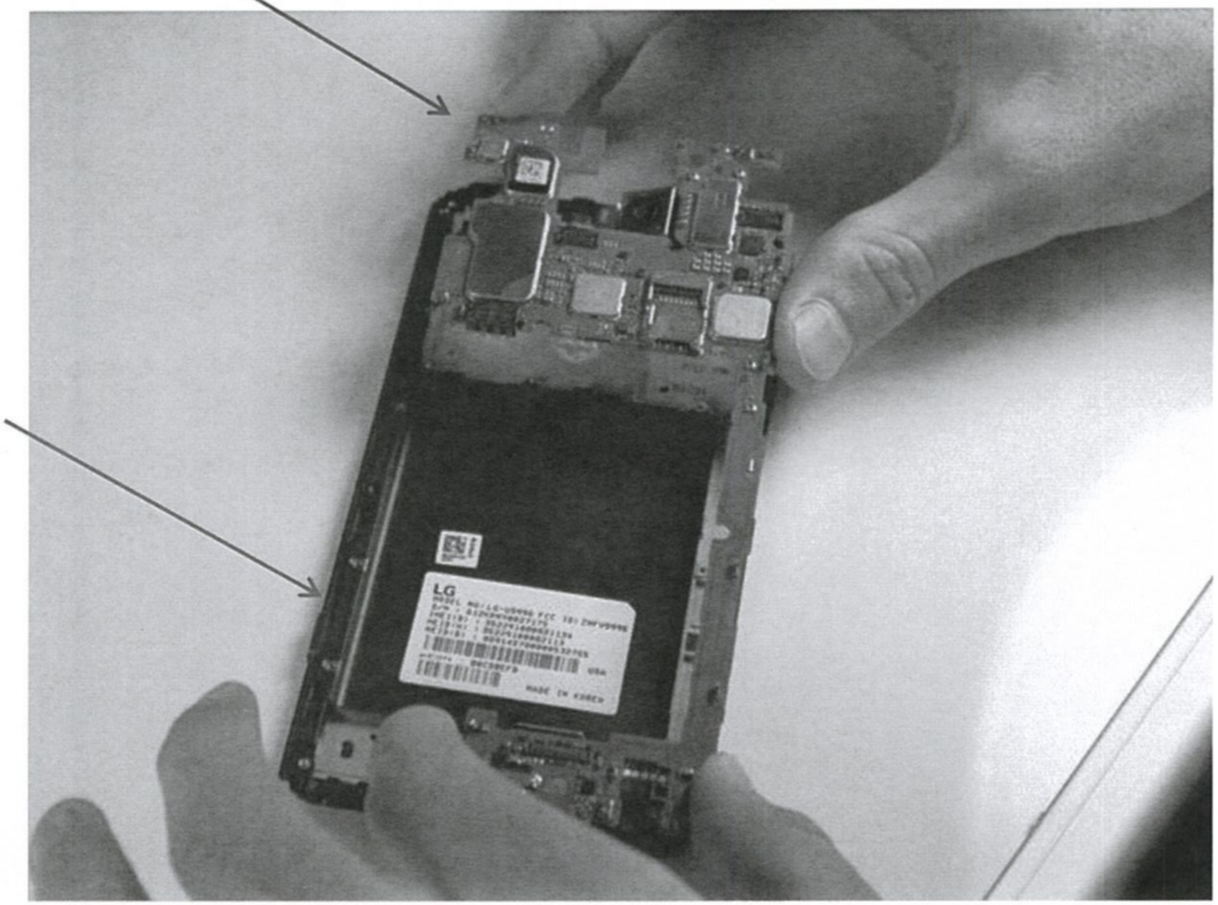
U.S. Patent No. 8,441,438 – LG V20

Claim 1
a **housing** associated with said movements and rotations of the 3D pointing device in a spatial pointer reference frame [*Samsung* Court's construction: frame of reference associated with the 3D pointing device, which always has its origin at the same point in the device and in which the axes are always fixed with respect to the device];

Source: http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

Claim 1
a **printed circuit board (PCB)** enclosed by the **housing**;

printed circuit board (PCB)

housing

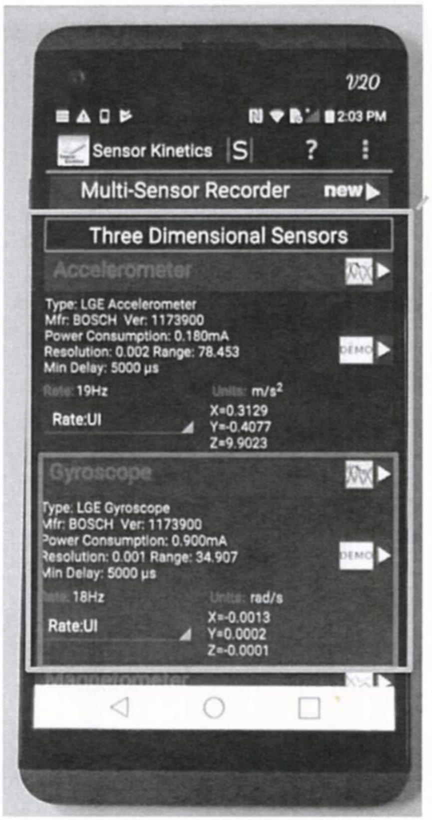


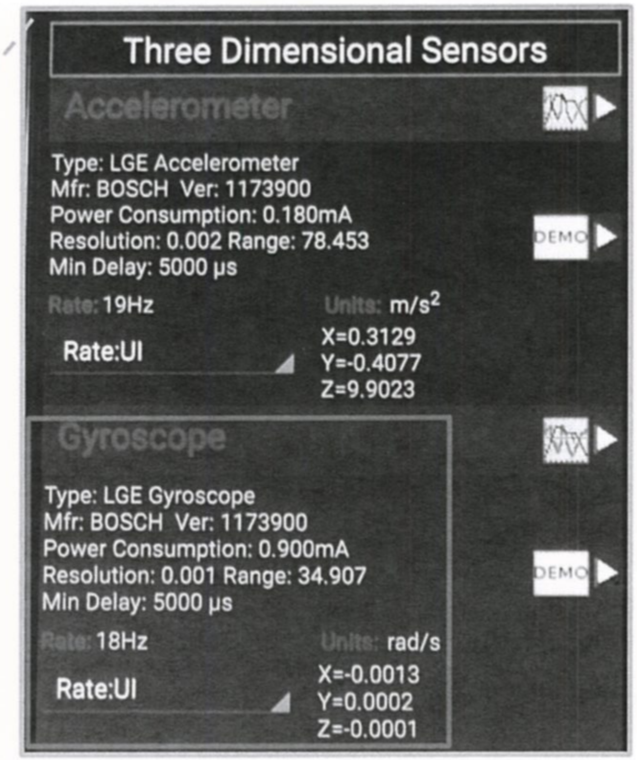
Source: https://www.ifixit.com/Guide/LG+V20+Motherboard+Assembly+Replacement/96392

Claim 1

a six-axis motion sensor module [*Samsung* Court's construction: no construction necessary] attached to the PCB, comprising a **rotation sensor** for detecting and generating a first signal set [*Samsung* Court's construction: no construction necessary] comprising angular velocities ω_x, ω_y, ω_z associated with said movements and rotations of the 3D pointing device in the spatial pointer reference frame,

The six-axis motion sensor module includes an accelerometer and gyroscope combo. The **rotation sensor** is the Gyroscope included in the six-axis motion sensor module.





Sensors Fingerprint (rear-mounted), accelerometer, gyro, proximity, compass, barometer, color spectrum

Source: https://www.gsmarena.com/lg_v20-8238.php

Claim 1
a six-axis motion sensor module [*Samsung* Court's construction: no construction necessary] attached to the PCB, comprising a **rotation sensor** for detecting and generating a **first signal set** [*Samsung* Court's construction: no construction necessary] comprising angular velocities ω_x, ω_y, ω_z associated with said movements and rotations of the 3D pointing device in the **spatial pointer reference frame**,

The **first signal set** includes the sensor event values of TYPE_GYROSCOPE.

Gyroscope

Reporting-mode: *Continuous*

`getDefaultSensor(SENSOR_TYPE_GYROSCOPE)` *returns a non-wake-up sensor*

A gyroscope sensor reports the rate of rotation of the device around the 3 sensor axes.

Rotation is positive in the counterclockwise direction (right-hand rule). That is, an observer looking from some positive location on the x, y or z axis at a device positioned on the origin would report positive rotation if the device appeared to be rotating counter clockwise. Note that this is the standard mathematical definition of positive rotation and does not agree with the aerospace definition of roll.

The measurement is reported in the x, y and z fields of `sensors_event_t.gyro` and all values are in radians per second (rad/s).

Source: https://source.android.com/devices/sensors/sensor-types#gyroscope

`Sensor.TYPE_GYROSCOPE:`

All values are in radians/second and measure the rate of rotation around the device's local X, Y and Z axis. The coordinate system is the same as is used for the acceleration sensor. Rotation is positive in the counter-clockwise direction. That is, an observer looking from some positive location on the x, y or z axis at a device positioned on the origin would report positive rotation if the device appeared to be rotating counter clockwise. Note that this is the standard mathematical definition of positive rotation and does not agree with the definition of roll given earlier.

- values[0]: Angular speed around the x-axis
- values[1]: Angular speed around the y-axis
- values[2]: Angular speed around the z-axis

Source: https://developer.android.com/reference/android/hardware/SensorEvent.html#values

Claim 1
a six-axis motion sensor module [*Samsung* Court's construction: no construction necessary] attached to the PCB, comprising a **rotation sensor** for detecting and generating a **first signal** set [*Samsung* Court's construction: no construction necessary] comprising angular velocities ω_x, ω_y, ω_z associated with said movements and rotations of the 3D pointing device in the spatial pointer reference frame,

Variable `w`, used by the `handleGyro()` function in the fusion.cpp file, represents gyroscope data or a **first signal set**.

```
void Fusion::handleGyro(const vec3_t& w, float dT) {
    if (!checkInitComplete(GYRO, w, dT))
        return;
```

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 1

a six-axis motion sensor module [*Samsung* Court's construction: no construction necessary] attached to the PCB, comprising a rotation sensor for detecting and generating a first signal set [*Samsung* Court's construction: no construction necessary] comprising angular velocities ω_x, ω_y, ω_z associated with said movements and rotations of the 3D pointing device in the **spatial pointer reference frame,**

Sensor Coordinate System

In general, the sensor framework uses a standard 3-axis coordinate system to express data values. For most sensors, the coordinate system is defined relative to the device's screen when the device is held in its default orientation (see figure 1). When a device is held in its default orientation, the X axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z axis points toward the outside of the screen face. In this system, coordinates behind the screen have negative Z values. This coordinate system is used by the following sensors:

- Acceleration sensor
- Gravity sensor
- Gyroscope
- Linear acceleration sensor
- Geomagnetic field sensor

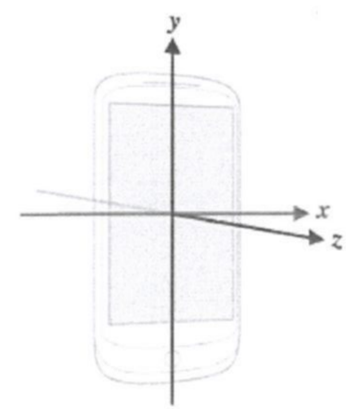


Figure 1. Coordinate system (relative to a device) that's used by the Sensor API.

The most important point to understand about this coordinate system is that the axes are not swapped when the device's screen orientation changes—that is, the sensor's coordinate system never changes as the device moves. This behavior is the same as the behavior of the OpenGL coordinate system.

Another point to understand is that your application must not assume that a device's natural (default) orientation is portrait. The natural orientation for many tablet devices is landscape. And the sensor coordinate system is always based on the natural orientation of a device.

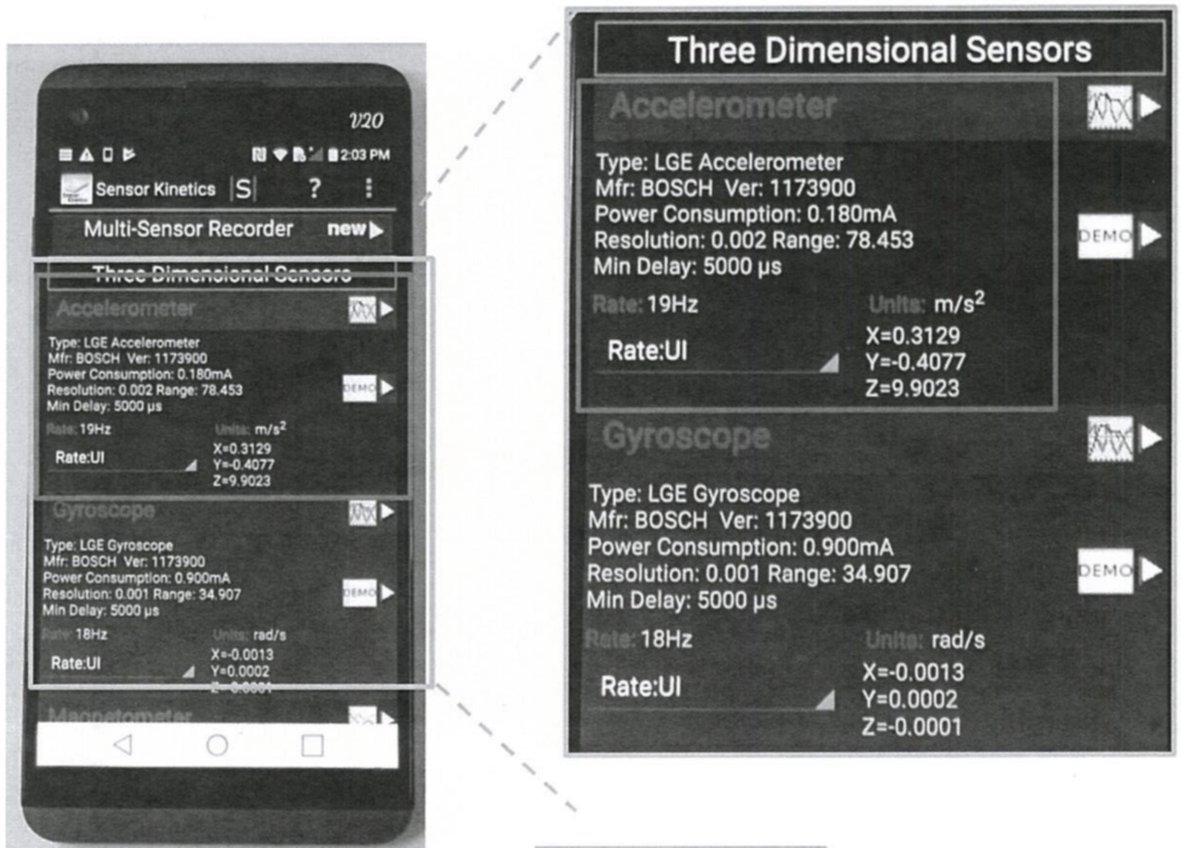
Source: http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

U.S. Patent No. 8,441,438 – LG V20

Claim 1

a six-axis motion sensor module [*Samsung* Court's construction: no construction necessary] attached to the PCB, comprising…an **accelerometer** for detecting and generating a second signal set [*Samsung* Court's construction: no construction necessary] comprising axial accelerations Ax, Ay, Az associated with said movements and rotations of the 3D pointing device in the spatial reference frame; and

The six-axis motion sensor module is an **accelerometer** and gyroscope combo.



Sensors Fingerprint (rear-mounted), accelerometer, gyro, proximity, compass, barometer, color spectrum

Source: https://www.gsmarena.com/lg_v20-8238.php

Claim 1

a **six-axis motion sensor module** [*Samsung* Court's construction: no construction necessary] attached to the PCB, comprising…an **accelerometer** for detecting and generating a **second signal set** [*Samsung* Court's construction: no construction necessary] comprising axial accelerations Ax, Ay, Az associated with said movements and rotations of the 3D pointing device in the spatial reference frame; and

The six-axis motion sensor module also includes an **accelerometer** for detecting and generating a **second signal set** comprising axial accelerations. The **second signal set** includes the sensor event values of TYPE_ACCELEROMETER.

Accelerometer

Reporting-mode: *Continuous*

`getDefaultSensor(SENSOR_TYPE_ACCELEROMETER)` *returns a non-wake-up sensor*

An accelerometer sensor reports the acceleration of the device along the 3 sensor axes. The measured acceleration includes both the physical acceleration (change of velocity) and the gravity. The measurement is reported in the x, y and z fields of sensors_event_t.acceleration.

All values are in SI units (m/s^2) and measure the acceleration of the device minus the force of gravity along the 3 sensor axes.

Source: https://source.android.com/devices/sensors/sensor-types#accelerometer

`Sensor.TYPE_ACCELEROMETER:`

All values are in SI units (m/s^2)

- values[0]: Acceleration minus Gx on the x-axis
- values[1]: Acceleration minus Gy on the y-axis
- values[2]: Acceleration minus Gz on the z-axis

A sensor of this type measures the acceleration applied to the device (**Ad**). Conceptually, it does so by measuring forces applied to the sensor itself (**Fs**) using the relation:

Ad = - ∑Fs / mass

In particular, the force of gravity is always influencing the measured acceleration:

Ad = -g - ∑F / mass

For this reason, when the device is sitting on a table (and obviously not accelerating), the accelerometer reads a magnitude of **g** = 9.81 m/s^2

Source: https://developer.android.com/reference/android/hardware/SensorEvent.html#values

Claim 1

a six-axis motion sensor module [*Samsung* Court's construction: no construction necessary] attached to the PCB, comprising…an accelerometer for detecting and generating a second signal set [*Samsung* Court's construction: no construction necessary] comprising axial accelerations Ax, Ay, Az associated with said movements and rotations of the 3D pointing device in the spatial reference frame; and

Variable a, used by the handleAcc() function in the fusion.cpp file, represents acceleration data or a **second signal set**.

```
status_t Fusion::handleAcc(const vec3_t& a, float dT) {
    if (!checkInitComplete(ACC, a, dT))
        return BAD_VALUE;
```

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 1

a six-axis motion sensor module [*Samsung* Court's construction: no construction necessary] attached to the PCB, comprising…an accelerometer for detecting and generating a second signal set [*Samsung* Court's construction: no construction necessary] comprising axial accelerations Ax, Ay, Az associated with said movements and rotations of the 3D pointing device in the **spatial reference frame**; and

Sensor Coordinate System

In general, the sensor framework uses a standard 3-axis coordinate system to express data values. For most sensors, the coordinate system is defined relative to the device's screen when the device is held in its default orientation (see figure 1). When a device is held in its default orientation, the X axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z axis points toward the outside of the screen face. In this system, coordinates behind the screen have negative Z values. This coordinate system is used by the following sensors:

- Acceleration sensor
- Gravity sensor
- Gyroscope
- Linear acceleration sensor
- Geomagnetic field sensor

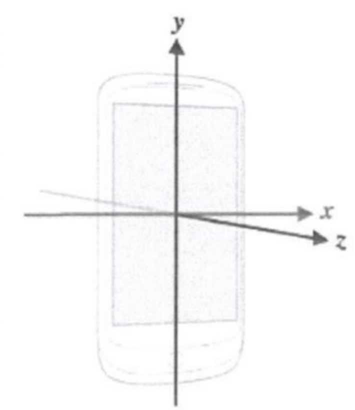


Figure 1. Coordinate system (relative to a device) that's used by the Sensor API.

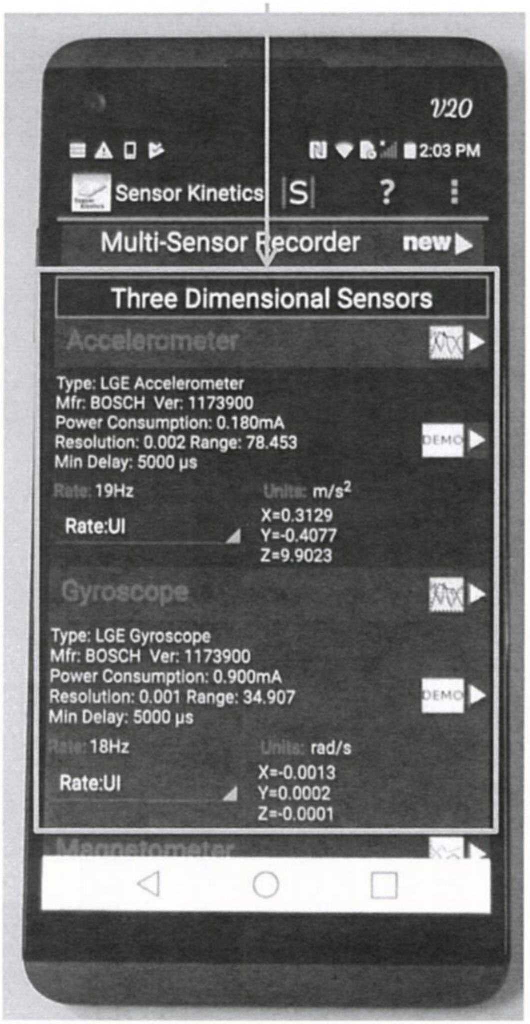
The most important point to understand about this coordinate system is that the axes are not swapped when the device's screen orientation changes—that is, the sensor's coordinate system never changes as the device moves. This behavior is the same as the behavior of the OpenGL coordinate system.

Another point to understand is that your application must not assume that a device's natural (default) orientation is portrait. The natural orientation for many tablet devices is landscape. And the sensor coordinate system is always based on the natural orientation of a device.

Source: http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

Claim 1
a processing and transmitting module, comprising a data transmitting unit electrically connected to the six-axis motion sensor module for transmitting said first and second signal sets thereof and a **computing processor** for <u>receiving and calculating said first and second signal sets from the data transmitting unit</u> [*Apple* Court's construction: no construction necessary].

six-axis motion sensor module



computing processor

Claim 1

communicating with the six-axis motion sensor module to calculate a **resulting deviation comprising resultant angles in said spatial pointer reference frame** [*Samsung* Court's construction: no construction necessary] by utilizing a comparison to compare the first signal set with the second signal set [*Apple* Court's Construction: using the calculation of actual deviation angles to compare the first signal set with the second signal set] whereby said resultant angles in the spatial pointer reference frame of the resulting deviation of the six-axis motion sensor module of the 3D pointing device are obtained under said dynamic environments,

Rotation vector

Underlying physical sensors: Accelerometer, Magnetometer, and Gyroscope

Reporting-mode: *Continuous*

`getDefaultSensor(SENSOR_TYPE_ROTATION_VECTOR)` *returns a non-wake-up sensor*

Source: https://source.android.com/devices/sensors/sensor-types#rotation_vector

getRotationMatrixFromVector

added in API level 9

```
void getRotationMatrixFromVector (float[] R,
                float[] rotationVector)
```

Helper function to convert a rotation vector to a rotation matrix. Given a rotation vector (presumably from a ROTATION_VECTOR sensor), returns a 9 or 16 element rotation matrix in the array R. R must have length 9 or 16. If R.length == 9, the following matrix is returned:

Source: https://developer.android.com/reference/android/hardware/SensorManager#getRotationMatrixFromVector(float[],%20float[])

getOrientation

added in API level 3

```
float[] getOrientation (float[] R,
                float[] values)
```

Computes the device's orientation based on the rotation matrix.

When it returns, the array values are as follows:

- values[0]: *Azimuth*, angle of rotation about the -z axis. This value represents the angle between the device's y axis and the magnetic north pole. When facing north, this angle is 0, when facing south, this angle is π. Likewise, when facing east, this angle is π/2, and when facing west, this angle is -π/2. The range of values is -π to π.
- values[1]: *Pitch*, angle of rotation about the x axis. This value represents the angle between a plane parallel to the device's screen and a plane parallel to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the top edge of the device toward the ground creates a positive pitch angle. The range of values is -π to π.
- values[2]: *Roll*, angle of rotation about the y axis. This value represents the angle between a plane perpendicular to the device's screen and a plane perpendicular to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the left edge of the device toward the ground creates a positive roll angle. The range of values is -π/2 to π/2.

Source: https://developer.android.com/reference/android/hardware/SensorManager#getOrientation(float[],%20float[])

Claim 1
communicating with the six-axis motion sensor module to calculate a resulting deviation comprising resultant angles in said spatial pointer reference frame [*Samsung* Court's construction: no construction necessary] by utilizing a comparison to compare the **first signal set** with the **second signal set** [*Apple* Court's Construction: using the calculation of actual deviation angles to compare the first signal set with the second signal set] whereby said resultant angles in the spatial pointer reference frame of the resulting deviation of the six-axis motion sensor module of the 3D pointing device are obtained under said dynamic environments,

The `predict()` function shows that the **first signal set** (angular velocities), w, is used to calculate the global variable x0.

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
```

```
    x0 = O*q;
```

The **second signal set** (axial accelerations) a, is passed to the variable z, and used in the `update()` function to update the global variable x0.

```
    vec3_t unityA = a * l_inv;
```

```
    update(unityA, Ba, p);
```

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
    // measured vector in body space: h(p) = A(p)*Bi
    const mat33_t A(quatToMatrix(q));
    const vec3_t Bb(A*Bi);
```

```
    const vec3_t e(z - Bb);
```

```
    x0 = normalize_quat(q);
```

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 1
communicating with the six-axis motion sensor module to calculate a resulting deviation comprising resultant angles in said spatial pointer reference frame [*Samsung* Court's construction: no construction necessary] by utilizing a **comparison to compare** **the first signal set** with the **second signal set** [*Apple* Court's Construction: using the calculation of actual deviation angles to compare the first signal set with the second signal set] whereby said resultant angles in the spatial pointer reference frame of the resulting deviation of the six-axis motion sensor module of the 3D pointing device are obtained under said dynamic environments,
The `predict()` function and `update()` functions are used in sensor fusion to update the global variable x0 in a quaternion form, which can represent actual deviation angles. In the `predict()` function, the **first signal set**, w, is used to calculate the global variable x0. In the `update()` function, x0 is converted to the variable Bb. The **second signal set**, a, is passed to the `update()` function as local variable z, and is used by the `update()` function to update the global variable x0. The variable Bb (from the **first signal set**) and the variable z (from the **second signal set**) are **compared** to calculate the variable e on line 529 of the Fusion.cpp file. Therefore, during the calculation of actual deviation angles, the first signal set is compared with the second signal set.

```
430   void Fusion::predict(const vec3_t& w, float dT) {
431       const vec4_t q  = x0;

485       x0 = O*q;

495   void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
496       vec4_t q(x0);
497       // measured vector in body space: h(p) = A(p)*Bi
498       const mat33_t A(quatToMatrix(q));
499       const vec3_t Bb(A*Bi);

529       const vec3_t e(z - Bb);

533       x0 = normalize_quat(q);
```

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 1
wherein the comparison utilized by the processing and transmitting module further comprises an update program to obtain an **updated state** based on a **previous state** associated with said **first signal set** and a **measured state** associated with said **second signal set**;
For example, the update program includes a `predict()` function and an `update()` function that are used to update the global variable x0 based on x0 (the **previous state**) associated with the **first signal set** w and e (the **measured state**) associated with the **second signal set** to calculate an **updated state** x0. The updated state x0 becomes the previous state x0 in the next iteration of the update program to obtain the updated state x0 in that iteration.

first signal set

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
```

previous state

```
    x0 = O*q;
```

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
```

measured state

second signal set

```
    const vec3_t e(z - Bb);
    const vec3_t dq(K[0]*e);

    q += getF(q)*(0.5f*dq);
    x0 = normalize_quat(q);
```

updated state

next iteration

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 1
wherein <u>**the measured state** includes **a measurement of said second signal set** and **a predicted measurement** obtained based on the first signal set **without using any derivatives of the first signal set** [*Samsung* Court's construction: the measured state includes a measurement of axial accelerations and predicted axial accelerations calculated using the angular velocities without computing derivatives of said angular velocities (i.e. angular accelerations)]</u>
The variable e is a **measured state** that includes a **measurement of said second signal set** z and a **predicted measurement** Bb calculated based on x0 (the previous state, which is calculated based on the first signal set).

second signal set (measured accelerations)

```
vec3_t unityA = a * l_inv;
```

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
    // measured vector in body space: h(p) = A(p)*Bi
    const mat33_t A(quatToMatrix(q));
    const vec3_t Bb(A*Bi);
```

```
const vec3_t e(z - Bb);
```

measured state **second signal set** **predicted measurement**

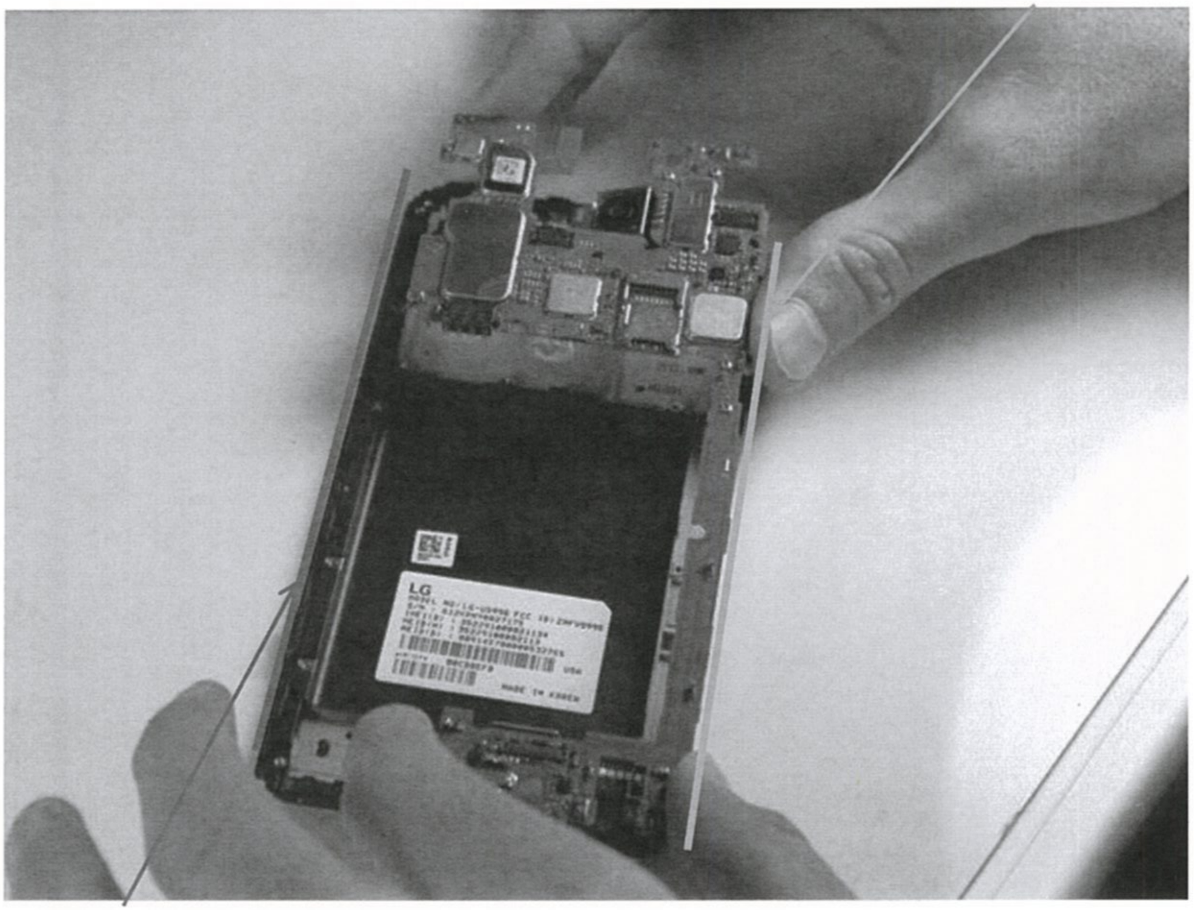
As shown in the code above, the predicted measurement is obtained based on the first signal set **without using any derivatives of the first signal set.**

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 3

The 3D pointing device of claim 1, wherein the PCB enclosed by the housing comprises at least one substrate having a **first longitudinal side** configured to be substantially parallel to a **longitudinal surface of the housing.**

first longitudinal side



longitudinal surface of the housing

Source: https://www.ifixit.com/Guide/LG+V20+Motherboard+Assembly+Replacement/96392

Claim 4

The 3D pointing device of claim 1, wherein the spatial pointer reference frame is a reference frame in three dimensions; and wherein said resultant angles of the resulting deviation includes **yaw, pitch and roll angles** about each of three orthogonal coordinate axes of the spatial pointer reference frame.

getOrientation

added in API level 3

```
float[] getOrientation (float[] R,
                float[] values)
```

Computes the device's orientation based on the rotation matrix.

When it returns, the array values are as follows:

- values[0]: *Azimuth*, angle of rotation about the -z axis. This value represents the angle between the device's y axis and the magnetic north pole. When facing north, this angle is 0, when facing south, this angle is π. Likewise, when facing east, this angle is π/2, and when facing west, this angle is -π/2. The range of values is -π to π.
- values[1]: *Pitch*, angle of rotation about the x axis. This value represents the angle between a plane parallel to the device's screen and a plane parallel to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the top edge of the device toward the ground creates a positive pitch angle. The range of values is -π to π.
- values[2]: *Roll*, angle of rotation about the y axis. This value represents the angle between a plane perpendicular to the device's screen and a plane perpendicular to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the left edge of the device toward the ground creates a positive roll angle. The range of values is -π/2 to π/2.

Source: https://developer.android.com/reference/android/hardware/SensorManager#getOrientation(float[], float[])

Sensor Coordinate System

In general, the sensor framework uses a standard 3-axis coordinate system to express data values. For most sensors, the coordinate system is defined relative to the device's screen when the device is held in its default orientation (see figure 1). When a device is held in its default orientation, the X axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z axis points toward the outside of the screen face. In this system, coordinates behind the screen have negative Z values. This coordinate system is used by the following sensors:

- Acceleration sensor
- Gravity sensor
- Gyroscope
- Linear acceleration sensor
- Geomagnetic field sensor

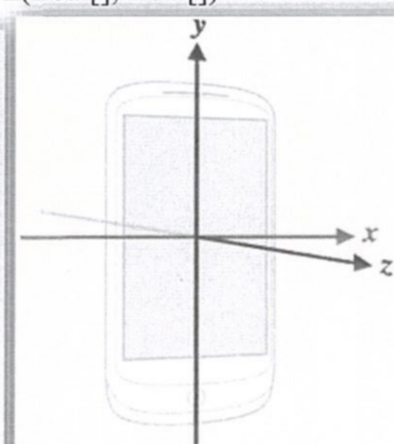


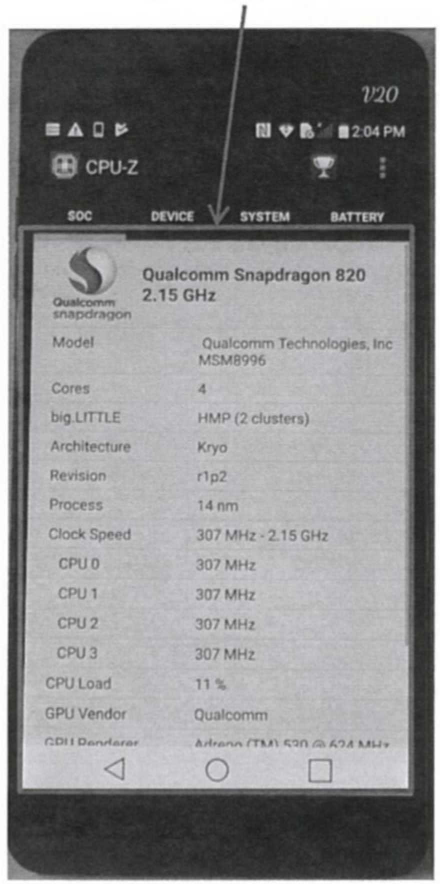
Figure 1. Coordinate system (relative to a device) that's used by the Sensor API.

Source: http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

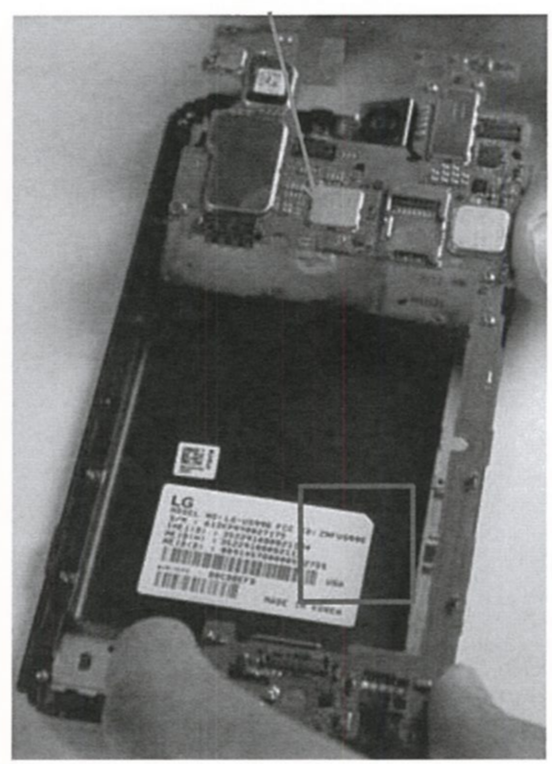
Claim 5
The 3D pointing device of claim 1, wherein the **data transmitting unit** of the processing and transmitting module is attached to the **PCB** enclosed by the housing and transmits said first and second signal of the **six-axis motion sensor module** to the **computer processor** via electronic connections.
The **computer processor** and the **six-axis motion sensor module** are each attached to the **PCB**, as is the **data transmitting unit**, which transmits the first and second signal of the **six-axis motion sensor module** to the **computer processor** via electronic connections.

six-axis motion sensor module

computing processor



PCB



Source:
https://www.ifixit.com/Guide/LG+V20+Motherboard+Assembly+Replacement/96392

Claim 14

A method for obtaining a resulting deviation including resultant angles in a **spatial pointer reference frame** of a **three-dimensional (3D) pointing device** utilizing a six-axis motion sensor module therein and subject to movements and rotations in dynamic environments in said **spatial pointer reference frame**, comprising the steps of:

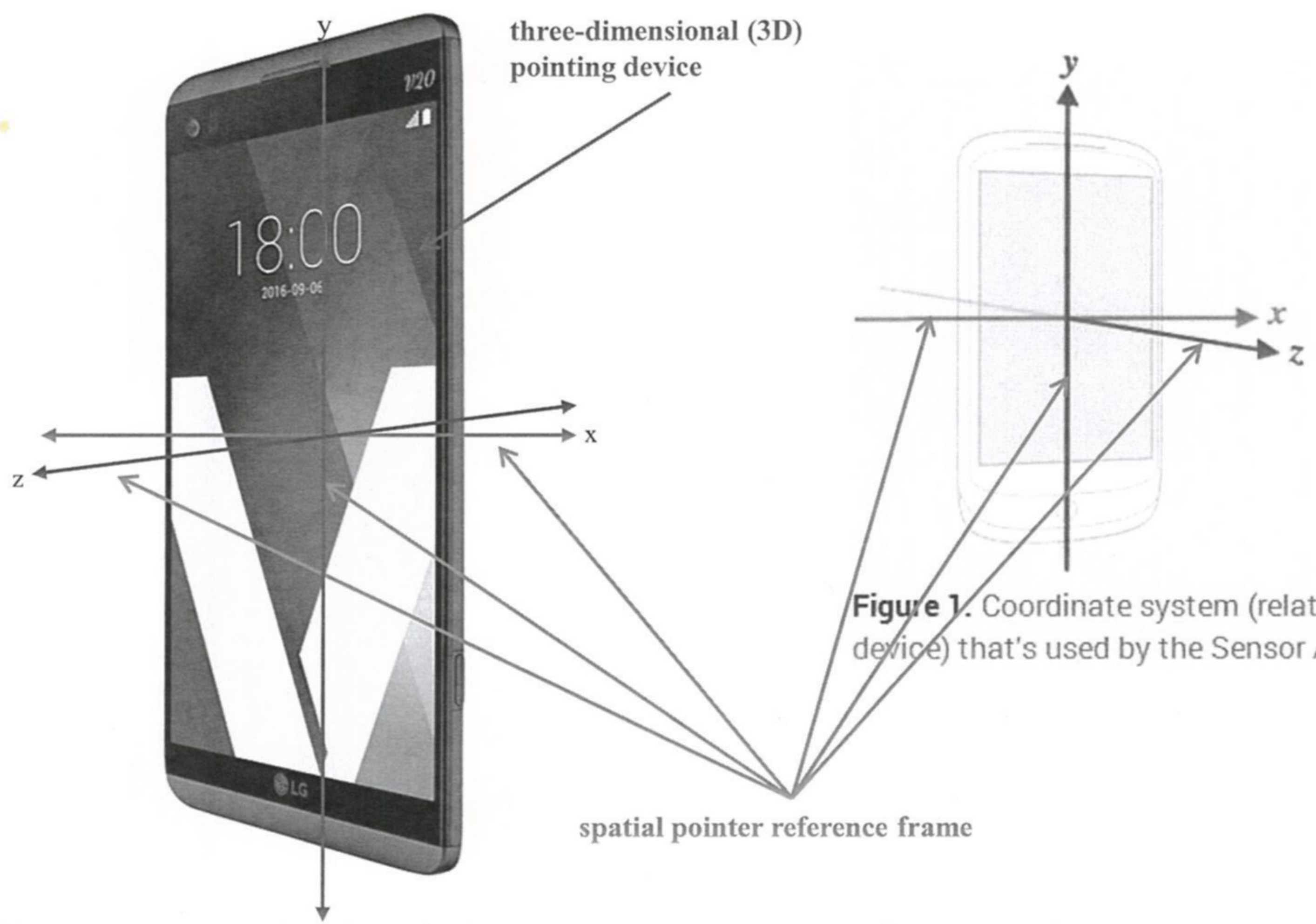


Figure 1. Coordinate system (relative to a device) that's used by the Sensor API.

Source: http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

Claim 14

obtaining a **previous state** of the six-axis motion sensor module; wherein the **previous state** includes an initial-value set associated with **previous angular velocities** gained from the motion sensor signals of the six-axis motion sensor module at a previous time T−1;

The previous state is obtained through an update program that includes a `predict()` function and an `update()` function. Those functions that are used to update the global variable x0 based on x0 (the **previous state**) associated with **previous angular velocities** w gained at a previous time T-1 to obtain an updated state x0. The updated state x0 becomes the previous state x0 at time T (the next iteration) of the update program to obtain the updated state x0 at time T.

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
```

← **previous state**

```
    x0 = O*q;
```

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
```

```
    const vec3_t e(z - Bb);
    const vec3_t dq(K[0]*e);

    q += getF(q)*(0.5f*dq);
    x0 = normalize_quat(q);
```

next iteration

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 14

obtaining a **current state** of the six-axis motion sensor module by obtaining **measured angular velocities** ω_x, ω_y, ω_z gained from the motion sensor signals of the six-axis motion sensor module at a current time T;

The `predict()` function runs during each iteration of the fusion algorithm, at a time T its output represents a **current state** output as x0. The `predict()` function is called by the `handleGyro()` function and receives **measured angular velocities**, w, associated with the **current state**.

```
void Fusion::handleGyro(const vec3_t& w, float dT) {
    if (!checkInitComplete(GYRO, w, dT))
        return;
```

measured angular velocities

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
```

```
    x0 = O*q;
```

current state

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 14

obtaining a **measured state** of the six-axis motion sensor module by obtaining **measured axial accelerations** Ax, Ay, Az gained from the motion sensor signals of the six-axis motion sensor module at the current time T and calculating **predicted axial accelerations** Ax′, Ay′, Az′ based on the **measured angular velocities** ω_x, ω_y, ω_z of the current state of the six-axis motion sensor module **without using any derivatives of the measured angular velocities** ωx, ωy, ωz;

The variable e is a **measured state** that includes **measured axil accelerations** z and **predicted axial accelerations** Bb calculated based on x0 (the previous state, which is calculated based on the **measured angular velocities**).

measured axial accelerations

```
        vec3_t unityA = a * l_inv;
```

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
    // measured vector in body space: h(p) = A(p)*Bi
    const mat33_t A(quatToMatrix(q));
    const vec3_t Bb(A*Bi);
```

```
        const vec3_t e(z - Bb);
```

measured state **measured axial accelerations** **predicted axial accelerations**

As shown in the code above, the predicted measurement is obtained based on the first signal set **without using any derivatives of the measured angular velocities.**

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 14

said **current state** of the six-axis motion sensor module is a second quaternion with respect to said current time T;

As shown in the examples provided, the **current state** is represented by the global state variable x0, which is a quaternion with respect to the current time T.

```
vec4_t Fusion::getAttitude() const {
    return x0;
}
```

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 14

comparing the second quaternion in relation to the **measured angular velocities** ω_x, ω_y, ω_z of the **current state** at current time T with the **measured axial accelerations** Ax, Ay, Az and the **predicted axial accelerations** Ax′, Ay′, Az′ also at current time T; obtaining an **updated state** of the six-axis motion sensor module by comparing the **current state** with the **measured state** of the six-axis motion sensor module; and

For example, as previously shown, the **measured state**, e, is obtained using the `update()` function, which combines the **measured axial accelerations**, z, and the **predicted axial accelerations**, Bb. Moreover, the **predicted axial accelerations** are determined based on the **measured angular velocities** of the **current state** at the current time T. The update() function further compares the **measured state**, e, and the **current state** to obtain the **updated state**, x0.

measured angular velocities

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
```

previous state

```
    x0 = O*q;
```

current state

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
```

measured state

measured axial accelerations

```
    const vec3_t e(z - Bb);
    const vec3_t dq(K[0]*e);

    q += getF(q)*(0.5f*dq);
    x0 = normalize_quat(q);
```

predicted axial accelerations

updated state

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 14
calculating and converting the **updated state** of the six axis motion sensor module to said **resulting deviation comprising said resultant angles** in said spatial pointer reference frame of the 3D pointing device.
The **updated state** x0 is in quaternion form, and can easily be converted to resultant angles. According to Android's developer library, the `getOrientation()` function "computes the device's orientation based on the rotation matrix," and returns **resultant angles** including the Azimuth, Pitch, and Roll angles.

getOrientation

Added in API level 3

```
float[] getOrientation (float[] R,
                float[] values)
```

Computes the device's orientation based on the rotation matrix.

When it returns, the array values are as follows:

- values[0]: *Azimuth*, angle of rotation about the -z axis. This value represents the angle between the device's y axis and the magnetic north pole. When facing north, this angle is 0, when facing south, this angle is π. Likewise, when facing east, this angle is π/2, and when facing west, this angle is -π/2. The range of values is -π to π.
- values[1]: *Pitch*, angle of rotation about the x axis. This value represents the angle between a plane parallel to the device's screen and a plane parallel to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the top edge of the device toward the ground creates a positive pitch angle. The range of values is -π to π.
- values[2]: *Roll*, angle of rotation about the y axis. This value represents the angle between a plane perpendicular to the device's screen and a plane perpendicular to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the left edge of the device toward the ground creates a positive roll angle. The range of values is -π/2 to π/2.

The `getRotationMatrixFromVector()` function "convert[s] a rotation vector to a rotation matrix," and the `getQuaternionFromVector()` function "convert[s] a rotation vector to a normalized quaternion." Therefore, the quaternion, x0, can be easily converted to its mathematically equivalent form, rotation matrix, and used by `getOrientation()` function to compute the orientation in its angular form.

Source: https://developer.android.com/reference/android/hardware/SensorManager#getOrientation(float[], float[])

Claim 15

The method for obtaining a resulting deviation of a 3D pointing device of claim 14, further comprises the step of outputting the **updated state** of the six-axis motion sensor module to the **previous state** of the six-axis motion sensor module; and wherein said resultant angles of the resulting deviation includes **yaw, pitch and roll angles** about each of three orthogonal coordinate axes of the spatial pointer reference frame.

For example, Android's source code discloses an iterative process for updating device motion. The **updated state** x0 output at time T-1 becomes an input of the **previous state** at time T and the "state" is iteratively updated.

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
```

previous state

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
```

updated state

```
x0 = normalize_quat(q);
```

next iteration

Moreover, the `getOrientation()` **function outputs yaw, pitch and roll angles.**

```
    public static float[] getOrientation(float[] R, float values[]) {
        if (R.length == 9) {
            values[0] = (float)Math.atan2(R[1], R[4]);
            values[1] = (float)Math.asin(-R[7]);
            values[2] = (float)Math.atan2(-R[6], R[8]);
        } else {
            values[0] = (float)Math.atan2(R[1], R[5]);
            values[1] = (float)Math.asin(-R[9]);
            values[2] = (float)Math.atan2(-R[8], R[10]);
        }
```

Source: https://android.googlesource.com/platform/frameworks/base/+/b267554/core/java/android/hardware/SensorManager.java

Claim 15

The method for obtaining a resulting deviation of a 3D pointing device of claim 14, further comprises the step of outputting the updated state of the six-axis motion sensor module to the previous state of the six-axis motion sensor module; and wherein said resultant angles of the resulting deviation includes **yaw, pitch and roll angles** about each of three orthogonal coordinate axes of the spatial pointer reference frame.

getOrientation

added in API level 3

```
float[] getOrientation (float[] R,
                float[] values)
```

Computes the device's orientation based on the rotation matrix.

When it returns, the array values are as follows:

- values[0]: *Azimuth, angle of rotation about the -z axis*. This value represents the angle between the device's y axis and the magnetic north pole. When facing north, this angle is 0, when facing south, this angle is π. Likewise, when facing east, this angle is π/2, and when facing west, this angle is -π/2. The range of values is -π to π.
- values[1]: *Pitch, angle of rotation about the x axis*. This value represents the angle between a plane parallel to the device's screen and a plane parallel to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the top edge of the device toward the ground creates a positive pitch angle. The range of values is -π to π.
- values[2]: *Roll, angle of rotation about the y axis*. This value represents the angle between a plane perpendicular to the device's screen and a plane perpendicular to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the left edge of the device toward the ground creates a positive roll angle. The range of values is -π/2 to π/2.

Source: https://developer.android.com/reference/android/hardware/SensorManager#getOrientation(float[], float[])

Sensor Coordinate System

In general, the sensor framework uses a standard 3-axis coordinate system to express data values. For most sensors, the coordinate system is defined relative to the device's screen when the device is held in its default orientation (see figure 1). When a device is held in its default orientation, the X axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z axis points toward the outside of the screen face. In this system, coordinates behind the screen have negative Z values. This coordinate system is used by the following sensors:

- Acceleration sensor
- Gravity sensor
- Gyroscope
- Linear acceleration sensor
- Geomagnetic field sensor

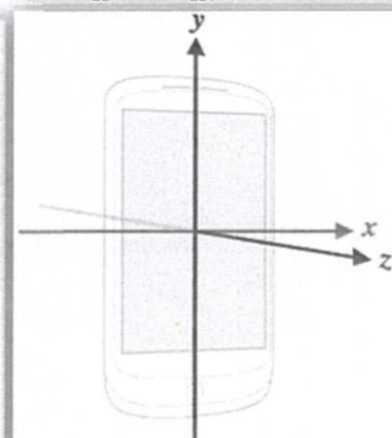


Figure 1. Coordinate system (relative to a device) that's used by the Sensor API.

Source: http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

Claim 16

The method for obtaining a resulting deviation of a 3D pointing device of claim 14, wherein said **previous state** of the six-axis motion sensor module is a **first quaternion** with respect to said previous time T−1; and said **updated state** of the six-axis motion sensor module is a **third quaternion** with respect to said current time T.

The **previous state** set by the `predict()` function takes the form of a **first quaternion**, x0.

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
```

previous state

The `update()` function calculates a **third quaternion** representing the **updated state**, x0.

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
```

```
        const vec3_t e(z - Bb);
        const vec3_t dq(K[0]*e);

        q += getF(q)*(0.5f*dq);
        x0 = normalize_quat(q);
```

updated state

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 17

The method for obtaining a resulting deviation of 3D pointing device of claim 14, wherein the obtaining of said previous state of the six-axis motion sensor module further comprises initializing said **initial-value set.**

The fusion algorithm sets an **initial-value set** as shown in the `initFusion()` function.

```
void Fusion::initFusion(const vec4_t& q, float dT)
{
    // initial estimate: E{ x(t0) }
    x0 = q;
    x1 = 0;
```

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 19
A method for obtaining a resulting deviation including resultant angles in a spatial pointer reference frame of a **three-dimensional (3D) pointing device** utilizing a six-axis motion sensor module therein and subject to movements and rotations in dynamic environments in said spatial pointer reference frame, comprising the steps of:

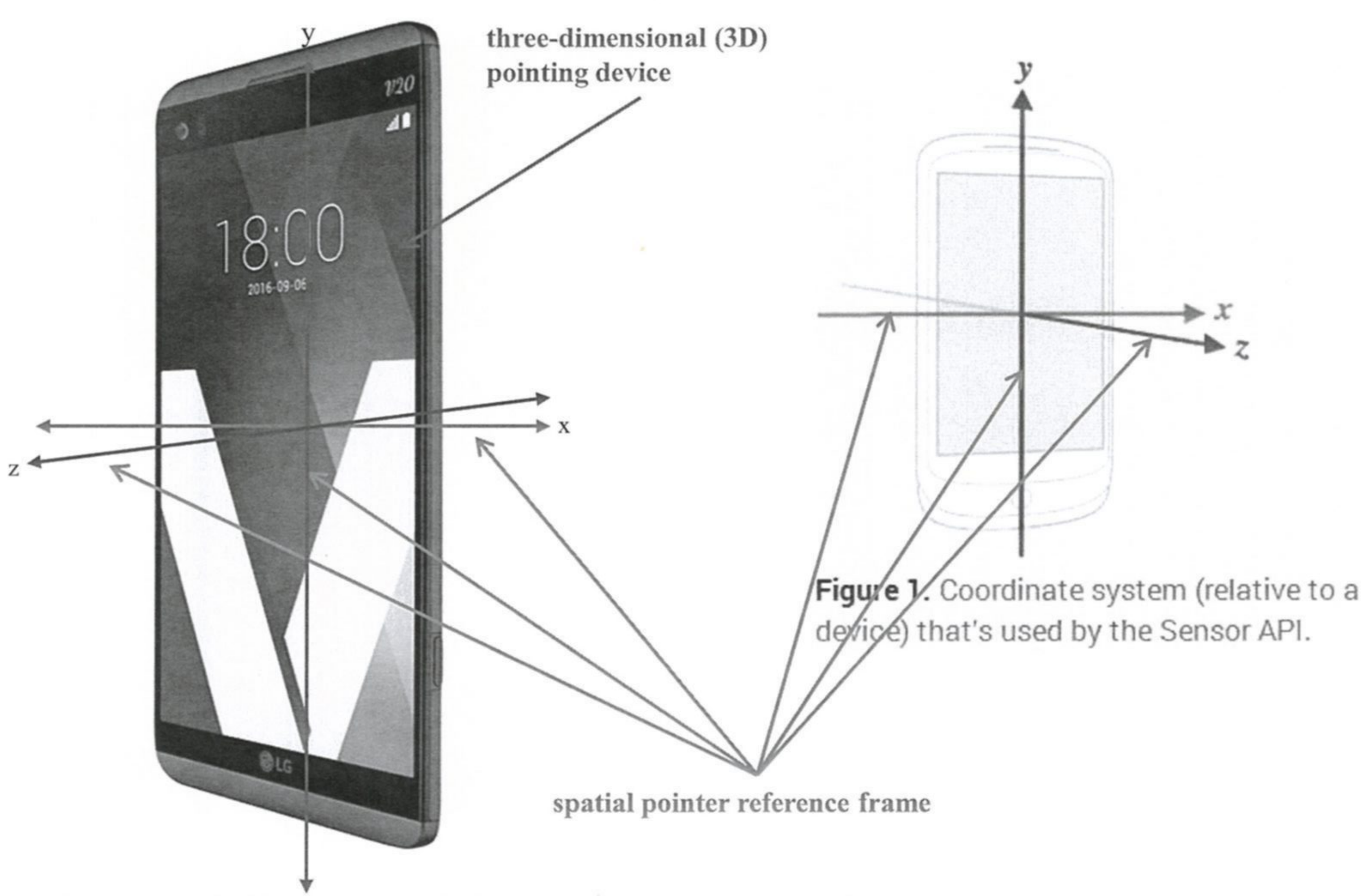


Figure 1. Coordinate system (relative to a device) that's used by the Sensor API.

Source: http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

Claim 19

obtaining a **previous state** of the six-axis motion sensor module; wherein the **previous state** includes an initial-value set associated with **previous angular velocities** gained from the motion sensor signals of the six-axis motion sensor module at a previous time T−1;

The previous state is obtained through an update program that includes a `predict()` function and an `update()` function. Those functions that are used to update the global variable x0 based on x0 (the **previous state**) associated with **previous angular velocities** w gained at a previous time T-1 to obtain an updated state x0. The updated state x0 becomes the previous state x0 at time T (the next iteration) of the update program to obtain the updated state x0 at time T.

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
```

previous state

```
    x0 = O*q;
```

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
```

```
    const vec3_t e(z - Bb);
    const vec3_t dq(K[0]*e);

    q += getF(q)*(0.5f*dq);
    x0 = normalize_quat(q);
```

next iteration

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 19

obtaining a **current state** of the six-axis motion sensor module by obtaining **measured angular velocities** ω_x, ω_y, ω_z gained from the motion sensor signals of the six-axis motion sensor module at a current time T;

The `predict()` function runs during each iteration of the fusion algorithm, at a time T its output represents a **current state** output as `x0`. The `predict()` function is called by the `handleGyro()` function and receives **measured angular velocities**, `w`, associated with the **current state**.

```
void Fusion::handleGyro(const vec3_t& w, float dT) {
    if (!checkInitComplete(GYRO, w, dT))
        return;
```

measured angular velocities

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
```

```
    x0 = O*q;
```

current state

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 19

obtaining a **measured state** of the six-axis motion sensor module by obtaining **measured axial accelerations** Ax, Ay, Az gained from the motion sensor signals of the six-axis motion sensor module at the current time T and calculating **predicted axial accelerations** Ax′, Ay′, Az′ based on the **measured angular velocities** ω_x, ω_y, ω_z of the current state of the six-axis motion sensor module **without using any derivatives of the measured angular velocities** ωx, ωy, ωz;

The variable e is a **measured state** that includes **measured axil accelerations** z and **predicted axial accelerations** Bb calculated based on x0 (the previous state, which is calculated based on the **measured angular velocities**).

measured axial accelerations

```
        vec3_t unityA = a * l_inv;
```

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
    // measured vector in body space: h(p) = A(p)*Bi
    const mat33_t A(quatToMatrix(q));
    const vec3_t Bb(A*Bi);
```

```
        const vec3_t e(z - Bb);
```

measured state **measured axial accelerations** **predicted axial accelerations**

As shown in the code above, the predicted measurement is obtained based on the first signal set **without using any derivatives of the measured angular velocities.**

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 19
said **current state** of the six-axis motion sensor module is a second quaternion with respect to said current time T;

As shown in the examples provided, the **current state** is represented by the global state variable x0, which is a quaternion with respect to the current time T.

```
vec4_t Fusion::getAttitude() const {
    return x0;
}
```

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 19
comparing the second quaternion in relation to the **measured angular velocities** ω_x, ω_y, ω_z of the **current state** at current time T with the **measured axial accelerations** Ax, Ay, Az and the **predicted axial accelerations** Ax′, Ay′, Az′ also at current time T; obtaining an **updated state** of the six-axis motion sensor module by comparing the **current state** with the **measured state** of the six-axis motion sensor module; and
For example, as previously shown, the **measured state**, e, is obtained using the update() function, which combines the **measured axial accelerations**, z, and the **predicted axial accelerations**, Bb. Moreover, the **predicted axial accelerations** are determined based on the **measured angular velocities** of the **current state** at the current time T. The update() function further compares the **measured state**, e, and the **current state** to obtain the **updated state**, x0.

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
...
    x0 = O*q;
...
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
...
    const vec3_t e(z - Bb);
    const vec3_t dq(K[0]*e);

    q += getF(q)*(0.5f*dq);
    x0 = normalize_quat(q);
```

measured angular velocities → w

previous state → x0 (line 431)

current state → x0 = O*q

measured axial accelerations → z

measured state → e

predicted axial accelerations → Bb

updated state → x0 = normalize_quat(q)

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 19
calculating and converting the **updated state** of the six axis motion sensor module to said **resulting deviation comprising said resultant angles** in said spatial pointer reference frame of the 3D pointing device.

The **updated state** `x0` is in quaternion form, and can easily be converted to resultant angles.

According to Android's developer library, the `getOrientation()` function "computes the device's orientation based on the rotation matrix," and returns **resultant angles** including the Azimuth, Pitch, and Roll angles.

getOrientation

Added in API level 3

```
float[] getOrientation (float[] R,
                float[] values)
```

Computes the device's orientation based on the rotation matrix.

When it returns, the array values are as follows:

- values[0]: *Azimuth*, angle of rotation about the -z axis. This value represents the angle between the device's y axis and the magnetic north pole. When facing north, this angle is 0, when facing south, this angle is π. Likewise, when facing east, this angle is π/2, and when facing west, this angle is -π/2. The range of values is -π to π.
- values[1]: *Pitch*, angle of rotation about the x axis. This value represents the angle between a plane parallel to the device's screen and a plane parallel to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the top edge of the device toward the ground creates a positive pitch angle. The range of values is -π to π.
- values[2]: *Roll*, angle of rotation about the y axis. This value represents the angle between a plane perpendicular to the device's screen and a plane perpendicular to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the left edge of the device toward the ground creates a positive roll angle. The range of values is -π/2 to π/2.

The `getRotationMatrixFromVector()` function "convert[s] a rotation vector to a rotation matrix," and the `getQuaternionFromVector()` function "convert[s] a rotation vector to a normalized quaternion." Therefore, the quaternion, `x0`, can be easily converted to its mathematically equivalent form, rotation matrix, and used by `getOrientation()` function to compute the orientation in its angular form.

Source: https://developer.android.com/reference/android/hardware/SensorManager#getOrientation(float[], float[])

EXHIBIT 14

CYWEE GROUP LTD,

vs.

LG ELECTRONICS, INC.,
LG ELECTRONICS U.S.A., INC.,
AND LG ELECTRONICS MOBILECOMM U.S.A., INC.,

UNITED STATES DISTRICT COURT
FOR THE SOUTHERN DISTRICT OF CALIFORNIA

EXEMPLARY CLAIM CHART

U.S. PATENT NO. 8,552,978 – LG V20
Infringement Contentions

These contentions are disclosed to only provide notice of Plaintiff's theories of infringement. These contentions do not constitute proof nor do they marshal Plaintiff's evidence of infringement to be presented during trial.

Claim 10

Claim 10 with claim constructions (text in brackets [] reflects the Court's claim construction or the parties' agreed claim construction in *CyWee Group, Ltd. v. Apple Inc.*, No. 3:13-cv-01853-HSG or in *CyWee Group, Ltd. v. Samsung Elecs. Co., Ltd.*, No. 2:17-cv-00140-WCB). Construed terms and constructions are underlined.

10. A method for compensating rotations of a 3D pointing device [*Samsung* Court's construction: no construction necessary], comprising:

generating an orientation output associated with an orientation of the 3D pointing device associated with three coordinate axes of a global reference frame associated with Earth [*Samsung* Court's construction: reference frame with axes defined with respect to the Earth];

generating a first signal set comprising axial accelerations associated with movements and rotations of the 3D pointing device in the spatial reference frame [*Samsung* Court's construction: frame of reference associated with the 3D pointing device, which always has its origin at the same point in the device and in which the axes are always fixed with respect to the device];

generating a second signal set associated with Earth's magnetism; generating the orientation output based on the first signal set, the second signal set and the rotation output or based on the first signal set and the second signal set;

generating a rotation output associated with a rotation of the 3D pointing device associated with three coordinate axes of a spatial reference frame associated with the 3D pointing device; and

using the orientation output and the rotation output to generate a transformed output associated with a fixed reference frame associated with a display device [*Apple* Court's construction: using the orientation output and the rotation output to generate a transformed output that corresponds to a two-dimensional movement in a plane that is parallel to the screen of a display device; *Samsung* Court's construction: using the orientation output and rotation output to generate a transformed output representing a movement in a fixed reference frame that is parallel to the screen of the display device], wherein the orientation output and the rotation output is generated by a nine-axis motion sensor module; obtaining one or more resultant deviation including a plurality of deviation angles using a plurality of measured magnetisms Mx, My, Mz and a plurality of predicted magnetism Mx′, My′ and Mz′ for the second signal set.

U.S. Patent No. 8,552,978 – LG V20

Claim 10
A method for compensating rotations of a 3D pointing device [*Samsung* Court's construction: no construction necessary], comprising:
 LG V20

Claim 10

generating an **orientation output** associated with an orientation of the 3D pointing device associated with three coordinate axes of a **global reference frame associated with Earth** [*Samsung* Court's construction: reference frame with axes defined with respect to the Earth];

When the orientation sensor is software-based, the **orientation output** is the attitude of the device that can be represented by the azimuth, pitch, and roll angles relative to the magnetic North Pole associated with a **global reference frame associated with Earth**.

Rotation vector

Underlying physical sensors: Accelerometer, Magnetometer, and Gyroscope

Reporting-mode: *Continuous*

`getDefaultSensor(SENSOR_TYPE_ROTATION_VECTOR)` *returns a non-wake-up sensor*

A rotation vector sensor reports the orientation of the device relative to the East-North-Up coordinates frame. It is usually obtained by integration of accelerometer, gyroscope, and magnetometer readings. The East-North-Up coordinate system is defined as a direct orthonormal basis where:

- X points east and is tangential to the ground.
- Y points north and is tangential to the ground.
- Z points towards the sky and is perpendicular to the ground.

The orientation of the phone is represented by the rotation necessary to align the East-North-Up coordinates with the phone's coordinates. That is, applying the rotation to the world frame (X,Y,Z) would align them with the phone coordinates (x,y,z).

Source: https://source.android.com/devices/sensors/sensor-types#rotation_vector

Claim 10

generating a **first signal set** comprising axial accelerations associated with movements and rotations of the 3D pointing device in the **spatial reference frame** [*Samsung* Court's construction: frame of reference associated with the 3D pointing device, which always has its origin at the same point in the device and in which the axes are always fixed with respect to the device];

Accelerometer

Reporting-mode: *Continuous*

`getDefaultSensor(SENSOR_TYPE_ACCELEROMETER)` *returns a non-wake-up sensor*

An accelerometer sensor reports the acceleration of the device along the 3 sensor axes. The measured acceleration includes both the physical acceleration (change of velocity) and the gravity. The measurement is reported in the x, y and z fields of sensors_event_t.acceleration.

All values are in SI units (m/s^2) and measure the acceleration of the device minus the force of gravity along the 3 sensor axes.

Source: https://source.android.com/devices/sensors/sensor-types#accelerometer

Sensor Coordinate System

In general, the sensor framework uses a standard 3-axis coordinate system to express data values. For most sensors, the coordinate system is defined relative to the device's screen when the device is held in its default orientation (see figure 1). When a device is held in its default orientation, the X axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z axis points toward the outside of the screen face. In this system, coordinates behind the screen have negative Z values. This coordinate system is used by the following sensors:

- Acceleration sensor
- Gravity sensor
- Gyroscope
- Linear acceleration sensor
- Geomagnetic field sensor

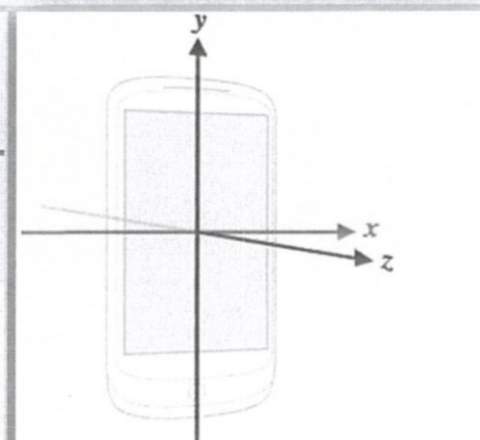


Figure 1. Coordinate system (relative to a device) that's used by the Sensor API.

Source: http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

Claim 10

generating a **second signal set** associated with **Earth's magnetism**;

The magnetometer (i.e., the compass) generates a second signal set associated with **Earth's magnetism**.

Magnetic field sensor

Reporting-mode: *Continuous*

`getDefaultSensor(SENSOR_TYPE_MAGNETIC_FIELD)` *returns a non-wake-up sensor*

`SENSOR_TYPE_GEOMAGNETIC_FIELD == SENSOR_TYPE_MAGNETIC_FIELD`

A magnetic field sensor (also known as magnetometer) reports the ambient magnetic field, as measured along the 3 sensor axes.

The measurement is reported in the x, y and z fields of `sensors_event_t.magnetic` and all values are in micro-Tesla (uT).

Source: https://source.android.com/devices/sensors/sensor-types#magnetic_field_sensor

Claim 10

generating the **orientation output** based on the **first signal set**, the **second signal set** and the **rotation output** or based on the **first signal set** and the **second signal set**;

The Android source code shows generating the **orientation output** based on the **first signal set**, the **second signal set** and the **rotation output**.

The `handleGyro()` function passes **rotation output** `w` to the `predict()` function and the `update()` function to calculate an **orientation output**, x0.

```
void Fusion::handleGyro(const vec3_t& w, float dT) {
    if (!checkInitComplete(GYRO, w, dT))
        return;
```

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
```

```
    x0 = O*q;
```

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
```

```
        const vec3_t e(z - Bb);
        const vec3_t dq(K[0]*e);

        q += getF(q)*(0.5f*dq);
        x0 = normalize_quat(q);
```

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 10

generating the **orientation output** based on the **first signal set**, the second signal set and the rotation output or based on the **first signal set** and the second signal set;

The handleAcc() function passes the accelerometer measurements (**first signal set**) a to the update() function, which updates the **orientation output** x0.

```
status_t Fusion::handleAcc(const vec3_t& a, float dT) {
    if (!checkInitComplete(ACC, a, dT))
        return BAD_VALUE;
```

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
```

```
        const vec3_t e(z - Bb);
        const vec3_t dq(K[0]*e);

        q += getF(q)*(0.5f*dq);
        x0 = normalize_quat(q);
```

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 10
generating the **orientation output** based on the first signal set, the **second signal set** and the rotation output or based on the first signal set and the **second signal set**;
The `handleMag()` function passes the magnetometer measurements (**second signal set**) `m` to the same `update()`, which also updates the **orientation output** `x0`.

```
status_t Fusion::handleMag(const vec3_t& m) {
    if (!checkInitComplete(MAG, m))
        return BAD_VALUE;
```

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
```

```
        const vec3_t e(z - Bb);
        const vec3_t dq(K[0]*e);

        q += getF(q)*(0.5f*dq);
        x0 = normalize_quat(q);
```

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 10

generating a **rotation output** associated with a rotation of the 3D pointing device associated with three coordinate axes of a **spatial reference frame** associated with the 3D pointing device; and

Gyroscope

Reporting-mode: *Continuous*

`getDefaultSensor(SENSOR_TYPE_GYROSCOPE)` *returns a non-wake-up sensor*

A gyroscope sensor reports the rate of rotation of the device around the 3 sensor axes.

Rotation is positive in the counterclockwise direction (right-hand rule). That is, an observer looking from some positive location on the x, y or z axis at a device positioned on the origin would report positive rotation if the device appeared to be rotating counter clockwise. Note that this is the standard mathematical definition of positive rotation and does not agree with the aerospace definition of roll.

Source: https://source.android.com/devices/sensors/sensor-types#gyroscope

Sensor Coordinate System

In general, the sensor framework uses a standard 3-axis coordinate system to express data values. For most sensors, the coordinate system is defined relative to the device's screen when the device is held in its default orientation (see figure 1). When a device is held in its default orientation, the X axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z axis points toward the outside of the screen face. In this system, coordinates behind the screen have negative Z values. This coordinate system is used by the following sensors:

- Acceleration sensor
- Gravity sensor
- Gyroscope
- Linear acceleration sensor
- Geomagnetic field sensor

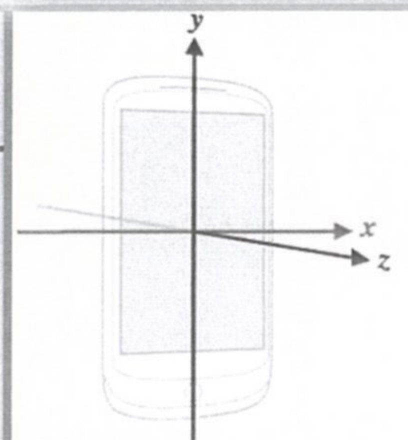


Figure 1. Coordinate system (relative to a device) that's used by the Sensor API.

Source: http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

Claim 10

using the orientation output and the rotation output to generate a transformed output associated with a **fixed reference frame** associated with a **display device** [*Apple* Court's construction: using the orientation output and the rotation output to generate a transformed output that corresponds to a two-dimensional movement in a plane that is parallel to the screen of a display device; *Samsung* Court's construction: using the orientation output and rotation output to generate a transformed output representing a movement in a fixed reference frame that is parallel to the screen of the display device],

The **fixed reference frame** is defined by the horizontal and vertical axes of pixels on the LG V20 **display device**.

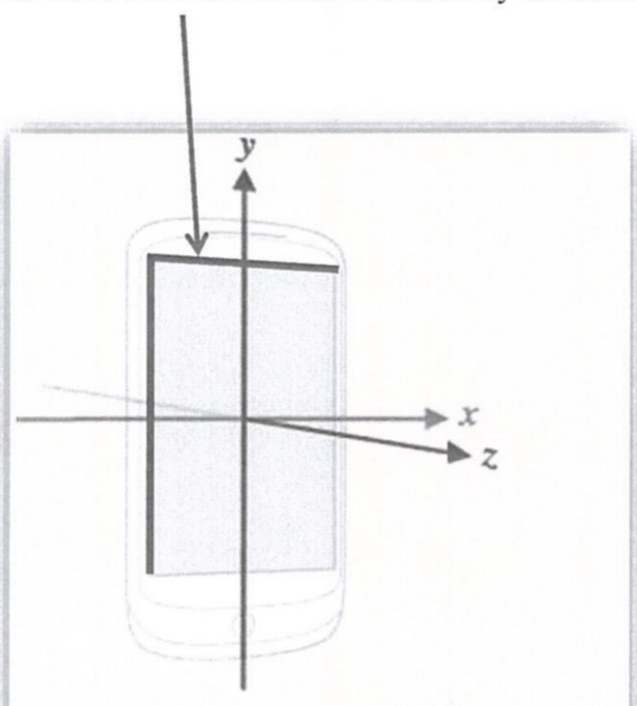
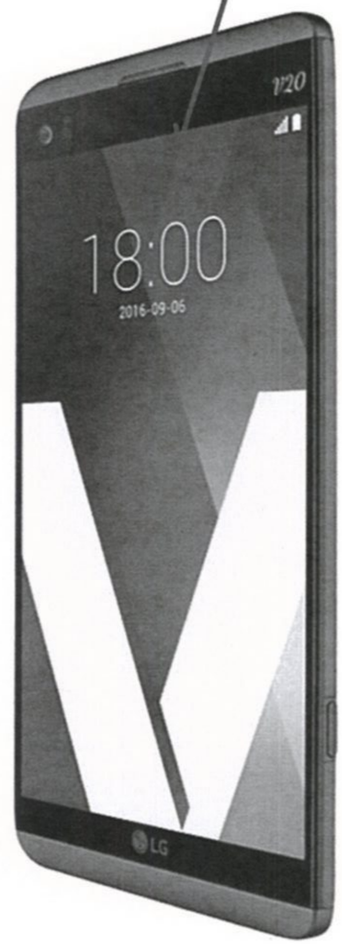


Figure 1. Coordinate system (relative to a device) that's used by the Sensor API.



Source:
http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

Claim 10

using the orientation output and the rotation output to generate a transformed output associated with a **fixed reference frame** associated with a **display device** [*Apple* Court's construction: using the orientation output and the rotation output to generate a transformed output that corresponds to a two-dimensional movement in a plane that is parallel to the screen of a display device; *Samsung* Court's construction: using the orientation output and rotation output to generate a transformed output representing a movement in a fixed reference frame that is parallel to the screen of the display device],

The `remapCoordinateSystem()` function transforms the orientation output (`inR`) to a transformed output (`outR`), associated with a two dimensional movement in a plane that is parallel to the screen of a **display device.**

```
public static boolean remapCoordinateSystem(float[] inR, int X, int Y, float[] outR) {
    if (inR == outR) {
        final float[] temp = sTempMatrix;
        synchronized (temp) {
            // we don't expect to have a lot of contention
            if (remapCoordinateSystemImpl(inR, X, Y, temp)) {
                final int size = outR.length;
                for (int i = 0; i < size; i++) {
                    outR[i] = temp[i];
                }
                return true;
            }
        }
    }
    return remapCoordinateSystemImpl(inR, X, Y, outR);
}
```

Source: https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/hardware/SensorManager.java

remapCoordinateSystem

added in API level 3

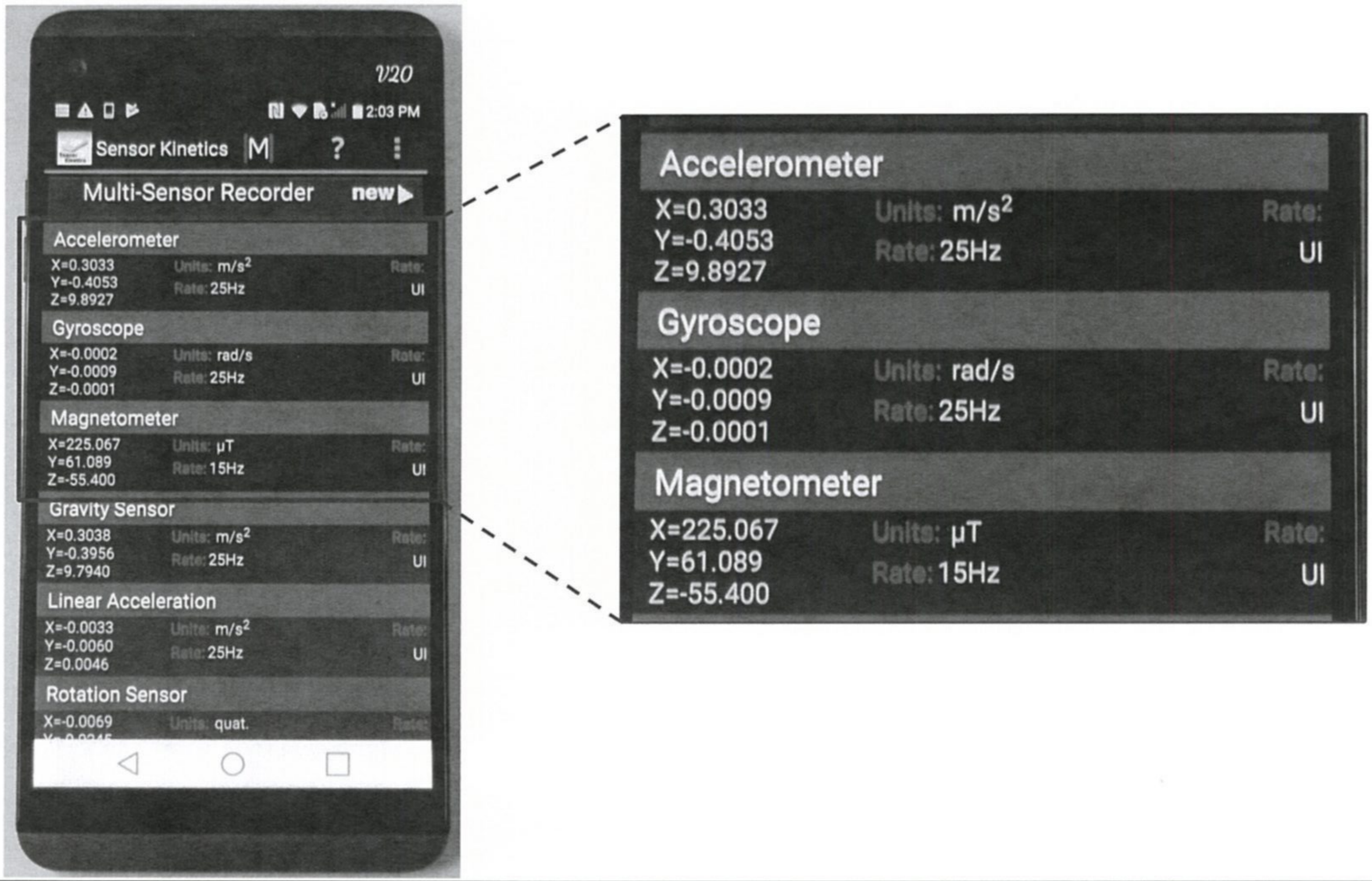
```
public static boolean remapCoordinateSystem (float[] inR,
                int X,
                int Y,
                float[] outR)
```

Rotates the supplied rotation matrix so it is expressed in a different coordinate system. This is typically used when an application needs to compute the three orientation angles of the device (see `getOrientation(float[], float[])`) in a different coordinate system.

When the rotation matrix is used for drawing (for instance with OpenGL ES), it usually **doesn't need** to be transformed by this function, unless the screen is physically rotated, in which case you can use `Display.getRotation()` to retrieve the current rotation of the screen. Note that because the user is generally free to rotate their screen, you often should consider the rotation in deciding the parameters to use here.

Source: https://developer.android.com/reference/android/hardware/SensorManager#remapCoordinateSystem(float[], int, int, float[])

Claim 10
wherein the orientation output and the rotation output is generated by a **nine-axis motion sensor module**;
The LG V20 includes a 3-axis gyroscope, a 3-axis accelerometer, and a 3-axis magnetometer which form a **nine-axis motion sensor module.**



Claim 10
obtaining one or more resultant deviation including a plurality of deviation angles using a plurality of **measured magnetisms Mx, My, Mz** and a plurality of predicted magnetism Mx′, My′ and Mz′ for the second signal set.

The **measured magnetisms Mx, My, Mz** are `values[0]-[2]`.

`Sensor.TYPE_MAGNETIC_FIELD_UNCALIBRATED`:

Similar to `TYPE_MAGNETIC_FIELD`, but the hard iron calibration is reported separately instead of being included in the measurement. Factory calibration and temperature compensation will still be applied to the "uncalibrated" measurement. Assumptions that the magnetic field is due to the Earth's poles is avoided.

The values array is shown below:

- values[0] = x_uncalib
- values[1] = y_uncalib
- values[2] = z_uncalib
- values[3] = x_bias
- values[4] = y_bias
- values[5] = z_bias

x_uncalib, y_uncalib, z_uncalib are the measured magnetic field in X, Y, Z axes. Soft iron and temperature calibrations are applied. But the hard iron calibration is not applied. The values are in micro-Tesla (uT).

x_bias, y_bias, z_bias give the iron bias estimated in X, Y, Z axes. Each field is a component of the estimated hard iron calibration. The values are in micro-Tesla (uT).

Hard iron - These distortions arise due to the magnetized iron, steel or permanenet magnets on the device. Soft iron - These distortions arise due to the interaction with the earth's magentic field.

Source: http://developer.android.com/reference/android/hardware/SensorEvent.html#values

Claim 10

obtaining one or more resultant deviation including a plurality of deviation angles using a plurality of **measured magnetisms Mx, My, Mz** and a plurality of **predicted magnetism** Mx′, My′ and Mz′ for the second signal set.

The **measured magnetisms**, z, and a **predicted magnetism**, Bb, are used to calculate a global variable x0 in quaternion form.

measured magnetisms

```
        update(m, Bm, mParam.magStdev);
```

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    const vec3_t Bb(A*Bi);
```

```
    const vec3_t e(z - Bb);
    const vec3_t dq(K[0]*e);

    q += getF(q)*(0.5f*dq);
    x0 = normalize_quat(q);
```

predicted magnetism

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 10
obtaining one or more **resultant deviation including a plurality of deviation angles** using a plurality of measured magnetisms Mx, My, Mz and a plurality of predicted magnetism Mx′, My′ and Mz′ for the second signal set. .

The global variable `x0` is in quaternion form, and can easily be converted to resultant angles.

According to Android's developer library, the `getOrientation()` function "computes the device's orientation based on the rotation matrix," and returns **deviation angles** including the Azimuth, Pitch, and Roll angles.

getOrientation

Added in API level 3

```
float[] getOrientation (float[] R,
                float[] values)
```

Computes the device's orientation based on the rotation matrix.

When it returns, the array values are as follows:

- values[0]: *Azimuth*, angle of rotation about the -z axis. This value represents the angle between the device's y axis and the magnetic north pole. When facing north, this angle is 0, when facing south, this angle is π. Likewise, when facing east, this angle is π/2, and when facing west, this angle is -π/2. The range of values is -π to π.
- values[1]: *Pitch*, angle of rotation about the x axis. This value represents the angle between a plane parallel to the device's screen and a plane parallel to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the top edge of the device toward the ground creates a positive pitch angle. The range of values is -π to π.
- values[2]: *Roll*, angle of rotation about the y axis. This value represents the angle between a plane perpendicular to the device's screen and a plane perpendicular to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the left edge of the device toward the ground creates a positive roll angle. The range of values is -π/2 to π/2.

The `getRotationMatrixFromVector()` **function "convert[s] a rotation vector to a rotation matrix," and the** `getQuaternionFromVector()` function "convert[s] a rotation vector to a normalized quaternion." Therefore, the quaternion, `x0`, can be easily converted to its mathematically equivalent form, rotation matrix, and used by `getOrientation()` function to compute the orientation in its angular form.

Source: https://developer.android.com/reference/android/hardware/SensorManager#getOrientation(float[], float[])

Claim 10

obtaining one or more **resultant deviation including a plurality of deviation angles** using a plurality of measured magnetisms Mx, My, Mz and a plurality of predicted magnetism Mx′, My′ and Mz′ for the second signal set. .

Rotation vector

Underlying physical sensors: Accelerometer, Magnetometer, and Gyroscope

Reporting-mode: *Continuous*

`getDefaultSensor(SENSOR_TYPE_ROTATION_VECTOR)` *returns a non-wake-up sensor*

Source: https://source.android.com/devices/sensors/sensor-types#rotation_vector

getRotationMatrixFromVector

added in API level 9

```
void getRotationMatrixFromVector (float[] R,
                float[] rotationVector)
```

Helper function to convert a rotation vector to a rotation matrix. Given a rotation vector (presumably from a ROTATION_VECTOR sensor), returns a 9 or 16 element rotation matrix in the array R. R must have length 9 or 16. If R.length == 9, the following matrix is returned:

Source: https://developer.android.com/reference/android/hardware/SensorManager#getRotationMatrixFromVector(float[], float[])

getOrientation

added in API level 3

```
float[] getOrientation (float[] R,
                float[] values)
```

Computes the device's orientation based on the rotation matrix.

When it returns, the array values are as follows:

- values[0]: *Azimuth*, angle of rotation about the -z axis. This value represents the angle between the device's y axis and the magnetic north pole. When facing north, this angle is 0, when facing south, this angle is π. Likewise, when facing east, this angle is π/2, and when facing west, this angle is -π/2. The range of values is -π to π.
- values[1]: *Pitch*, angle of rotation about the x axis. This value represents the angle between a plane parallel to the device's screen and a plane parallel to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the top edge of the device toward the ground creates a positive pitch angle. The range of values is -π to π.
- values[2]: *Roll*, angle of rotation about the y axis. This value represents the angle between a plane perpendicular to the device's screen and a plane perpendicular to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the left edge of the device toward the ground creates a positive roll angle. The range of values is -π/2 to π/2.

Source: https://developer.android.com/reference/android/hardware/SensorManager#getOrientation(float[], float[])

Claim 12

The method of claim 10, wherein the **orientation output** is a **rotation matrix**, a quaternion, a rotation vector, or comprises three orientation angles.

Several methods are provided for receiving the **orientation output** as a rotation matrix, a quaternion, or three orientation angles. The `getRotationMatrix()` function outputs a **rotation matrix**.

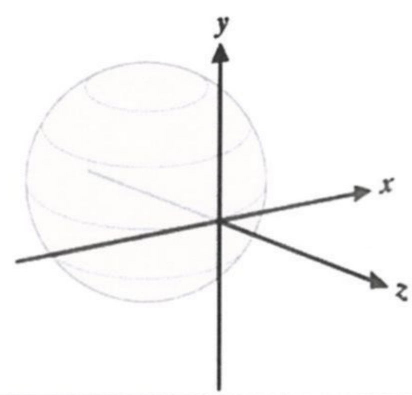
getRotationMatrix

added in API level 3

```
boolean getRotationMatrix (float[] R,
                float[] I,
                float[] gravity,
                float[] geomagnetic)
```

Computes the inclination matrix **I** as well as the rotation matrix **R** transforming a vector from the device coordinate system to the world's coordinate system which is defined as a direct orthonormal basis, where:

- X is defined as the vector product **Y.Z** (It is tangential to the ground at the device's current location and roughly points East).
- Y is tangential to the ground at the device's current location and points towards the magnetic North Pole.
- Z points towards the sky and is perpendicular to the ground.



Source: https://developer.android.com/reference/android/hardware/SensorManager#getRotationMatrix(float[], float[], float[], float[])

Claim 12

The method of claim 10, wherein the orientation output is a rotation matrix, a **quaternion**, a rotation vector, or comprises **three orientation angles**.

The `getQuaternionFromVector()` function outputs a **quaternion**.

`getQuaternionFromVector(float[] Q, float[] rv)`

Helper function to convert a rotation vector to a normalized quaternion.

The `getOrientation()` function outputs **three orientation angles**.

getOrientation

Added in API level 3

```
float[] getOrientation (float[] R,
                float[] values)
```

Computes the device's orientation based on the rotation matrix.

When it returns, the array values are as follows:

- values[0]: *Azimuth*, angle of rotation about the -z axis. This value represents the angle between the device's y axis and the magnetic north pole. When facing north, this angle is 0, when facing south, this angle is π. Likewise, when facing east, this angle is π/2, and when facing west, this angle is -π/2. The range of values is -π to π.
- values[1]: *Pitch*, angle of rotation about the x axis. This value represents the angle between a plane parallel to the device's screen and a plane parallel to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the top edge of the device toward the ground creates a positive pitch angle. The range of values is -π to π.
- values[2]: *Roll*, angle of rotation about the y axis. This value represents the angle between a plane perpendicular to the device's screen and a plane perpendicular to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the left edge of the device toward the ground creates a positive roll angle. The range of values is -π/2 to π/2.

Source: https://developer.android.com/reference/android/hardware/SensorManager#getOrientation(float[], float[])