Introduction to Android Virtual Sensor

CS 436 Software Development on Mobile

Dr.Paween Khoenkaw

Department of Computer Science
Maejo University

Virtual sensor

Problem of using raw data

- -Noise
  - -RF signals
  - -Electrical noise
  - -Magnetic anomalies
  - -Temperature variations
  - -Shock and vibration
- -Signal processing
  - -Real-time tasks required
  - -Unreliable time stamps of sensor samples
  - -Low-pass filtering can discard useful information

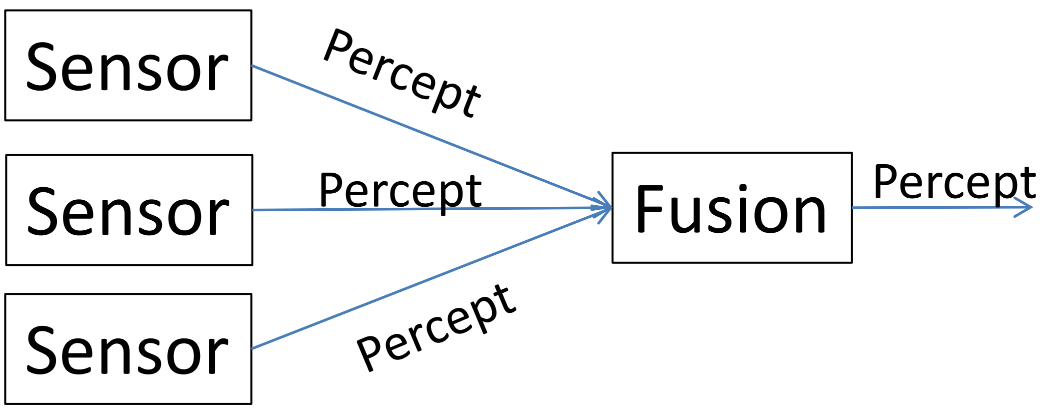
Virtual sensor

The solution is Sensor fusion

Using data from multiple sensor to create new virtual sensor

Virtual sensor

Virtual sensor is <u>software</u> that bridge what can be measured to what developers want to detect



Virtual sensor

Sensor fusion can be implement by

-Driver
-Kernel
-Co-processor

Virtual sensor

TYPE_GRAVITY
TYPE_LINEAR_ACCELERATION
TYPE_ROTATION_VECTOR
TYPE_ORIENTATION

Virtual sensor

Accelerometer = **Gravity**

Accelerometer- **Gravity** = **Linear Acceleration**

Compass + Accelerometer = **Orientation**

Compass + Accelerometer+Gyroscope = **Rotation**

Virtual sensor

Gravity

Gravity sensor is de-noise version of accelerometer

Unit : m/s^2

Virtual sensor

Linear Acceleration

Accelerometer- **Gravity** = **Linear Acceleration**

Unit : m/s^2

Virtual sensor

Orientation

Compass + Accelerometer = **Orientation**

Output value based on Euler Angles

values[0]: Azimuth, angle between the magnetic north direction and the y-axis, around the **z-axis** (0 to 359). 0=North, 90=East, 180=South, 270=West

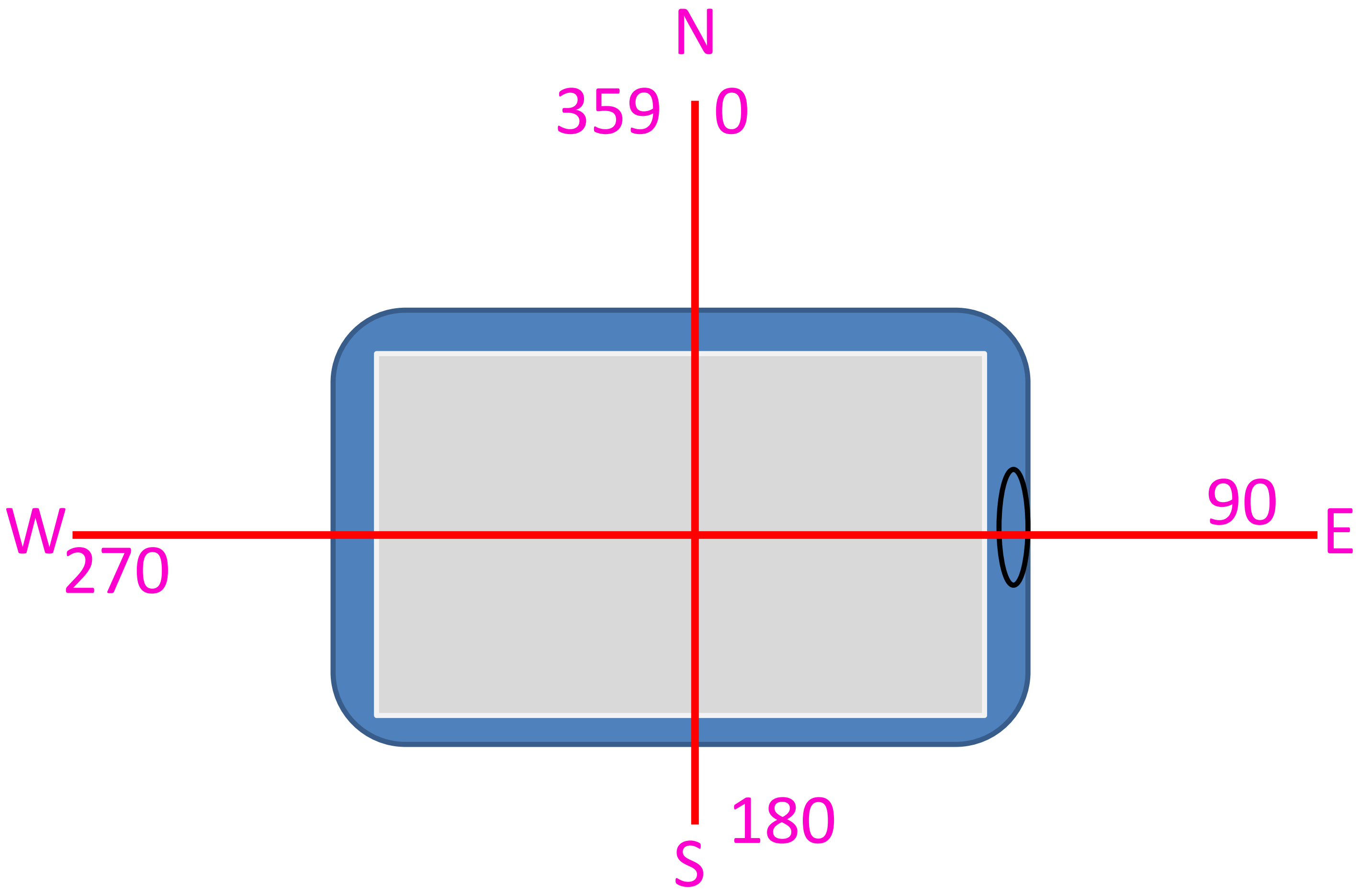
values[1]: Pitch, rotation around x-axis (-180 to 180), with positive values when the z-axis moves toward the **y-axis**.

values[2]: Roll, rotation around y-axis (-90 to 90), with positive values when the x-axis moves toward the **z-axis.**

Virtual sensor

Orientation

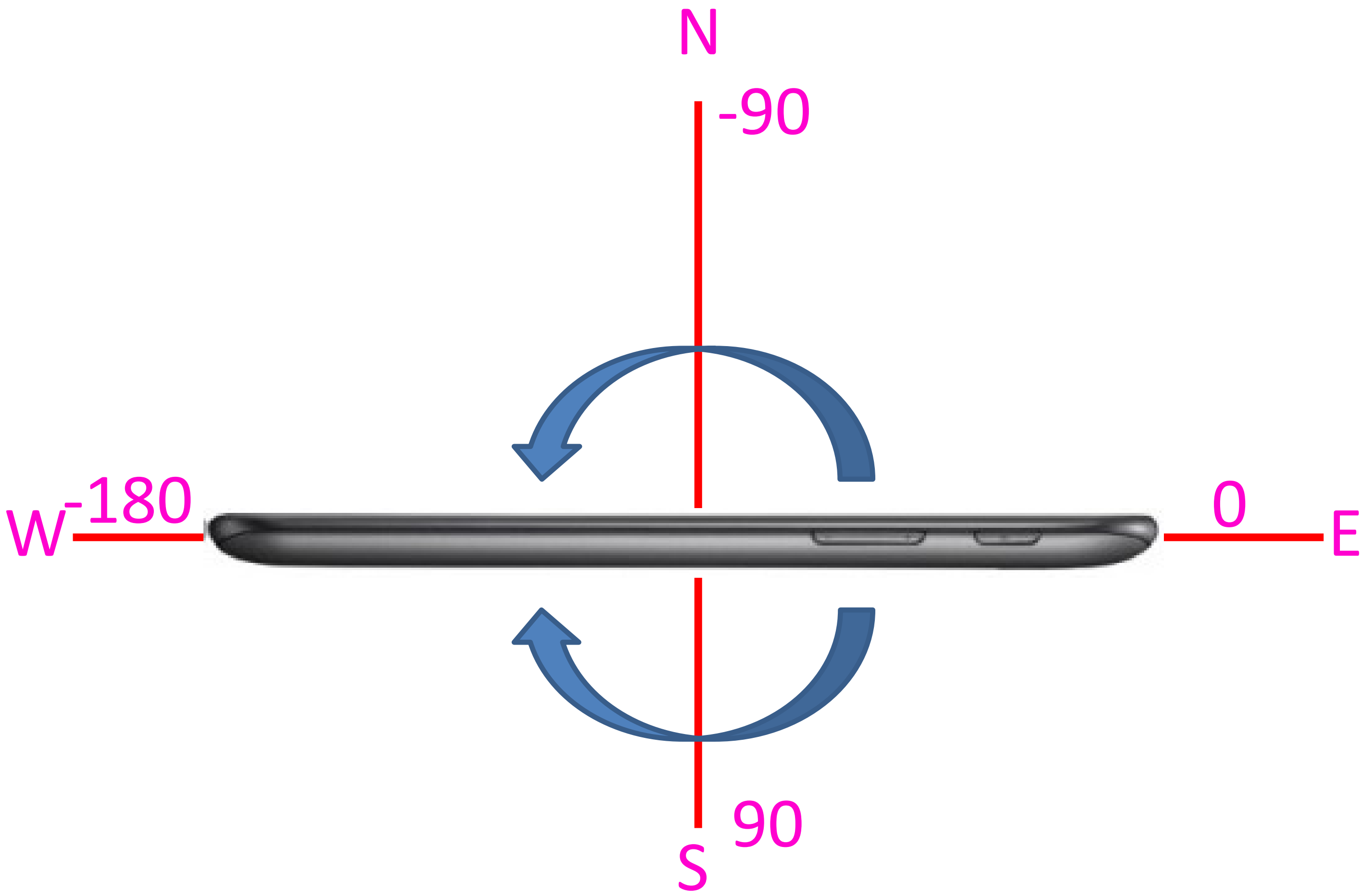
values[0]: Azimuth



Virtual sensor

Orientation

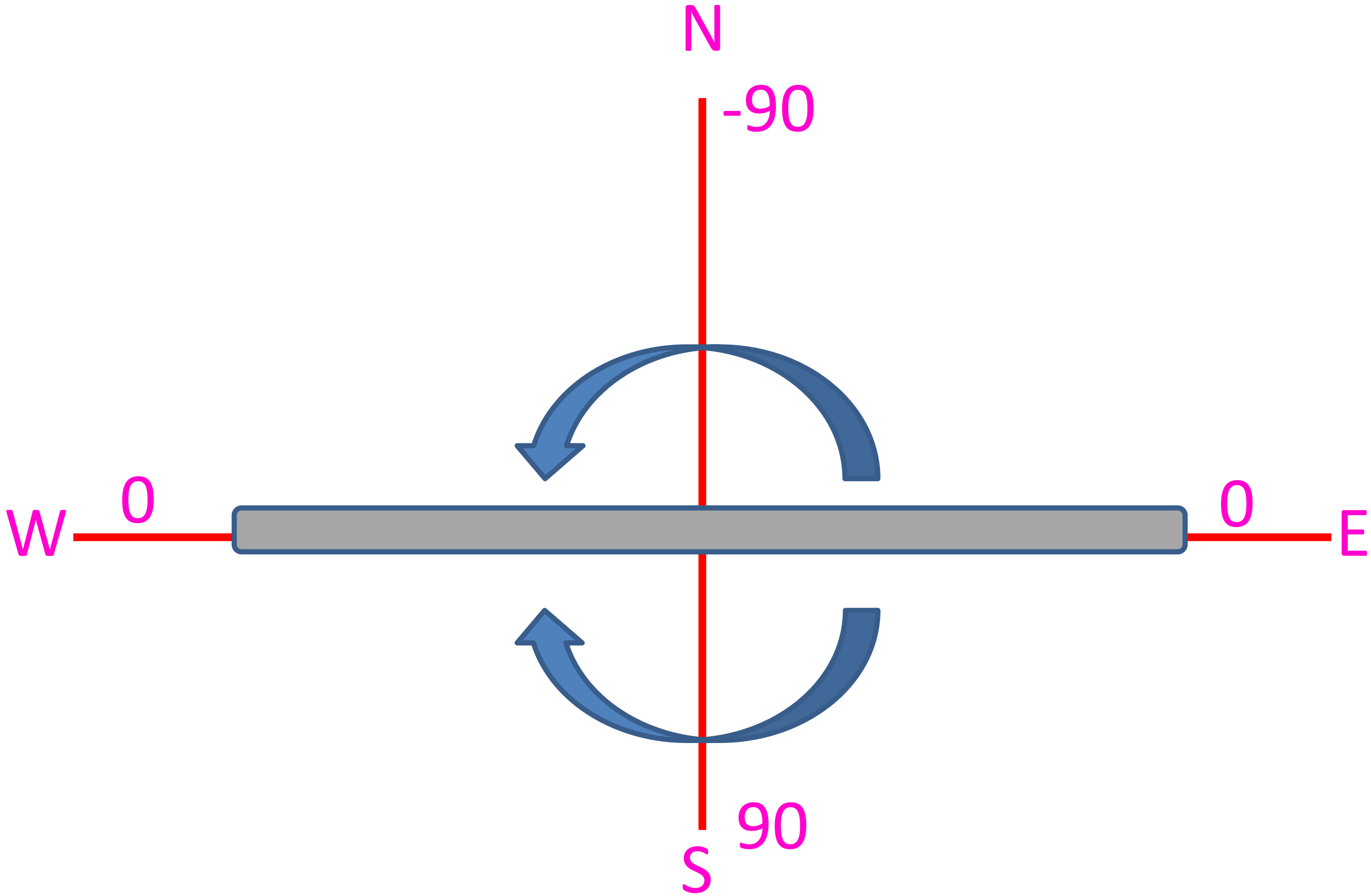
values[1]: Pitch



Virtual sensor

Orientation

values[2]: Roll



Virtual sensor

Orientation

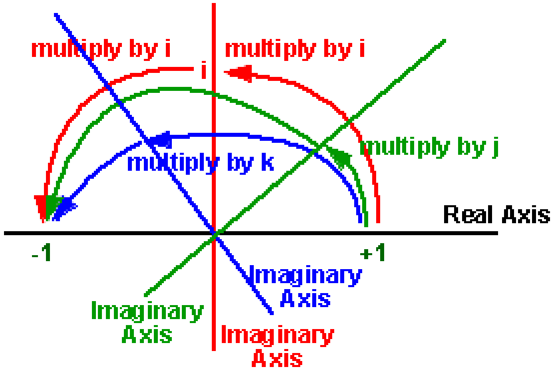
- This sensor type exists for legacy reasons
- For historical reasons the roll angle is positive in the clockwise direction

Virtual sensor

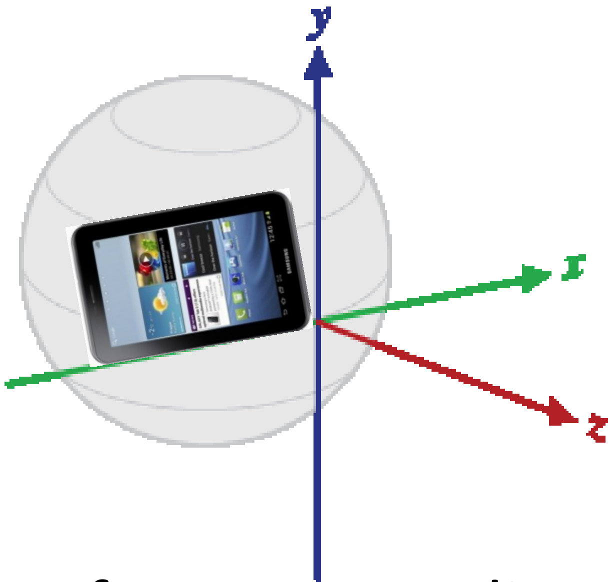
Rotation

Compass + Accelerometer+Gyroscope = **Rotation**

Return result as Quaternion



North



Reference coordinate

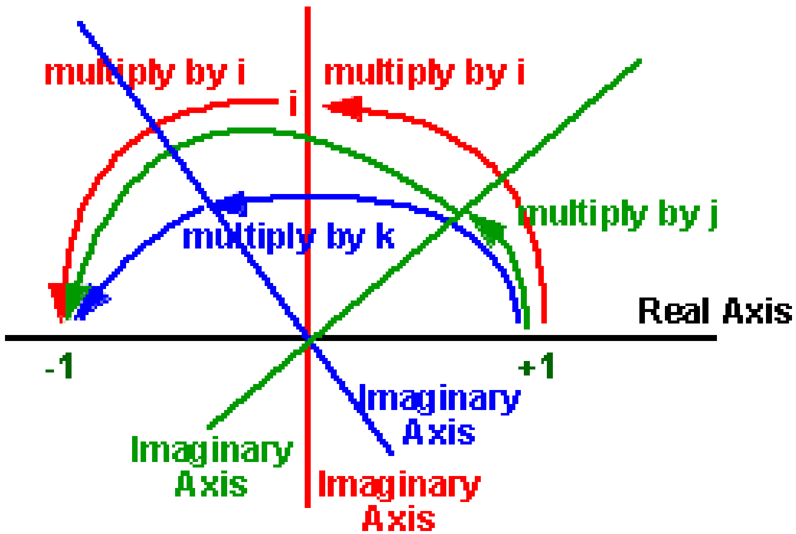
Virtual sensor

Rotation

Compass + Accelerometer+Gyroscope = **Rotation**

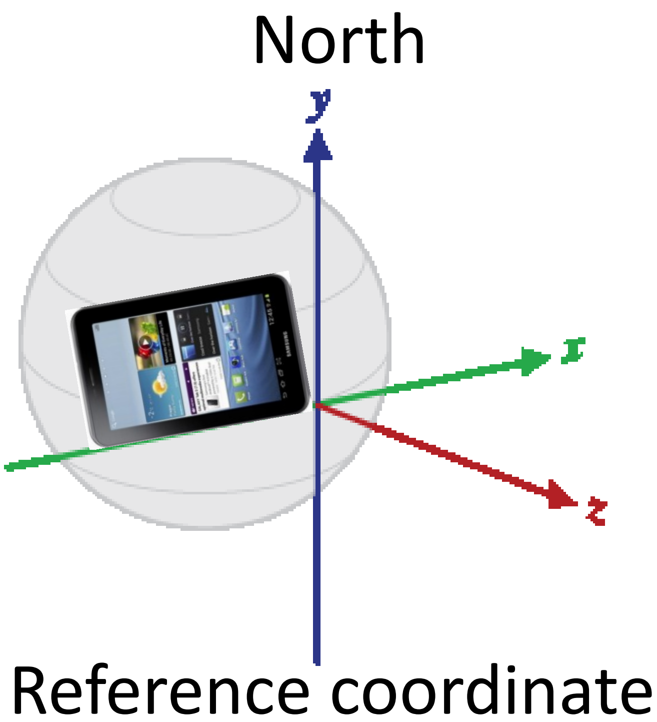
Return result as Quaternion

[w + i b + j c + k d]



[cos(θ/2), x*sin(θ/2), y*sin(θ/2), z*sin(θ/2)]

Virtual sensor



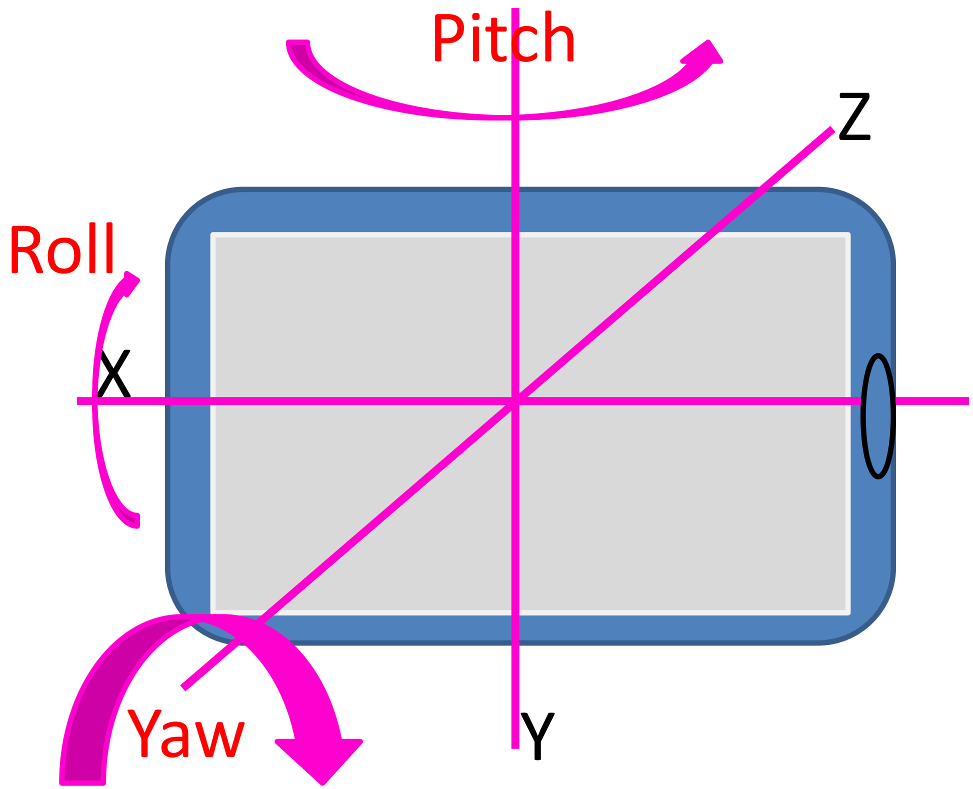
$\Theta_x=0$
$\Theta_y=0$
$\Theta_z=0$

w= cos(θ/2)=1
x=x*sin(θ/2)=0
y=y*sin(θ/2)=0
z=z*sin(θ/2) =0

Q=[1 0 0 0]

Virtual sensor

Euler angle to Quaternion conversion

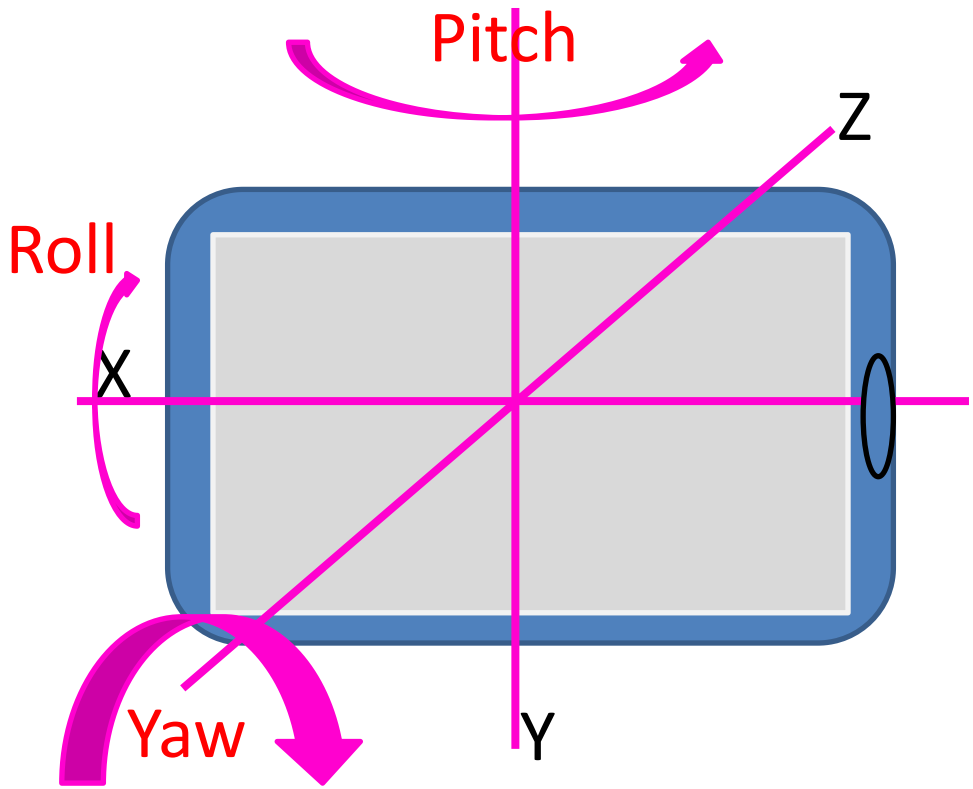



```
c1 = cos(pitch / 2);
c2 = cos(yaw / 2);
c3 = cos(roll / 2);
s1 = sin(pitch / 2);
s2 = sin(yaw / 2);
s3 = sin(roll / 2);

w = (c1* c2* c3) - (s1* s2 *s3);
x = (s1* s2* c3) +(c1 *c2* s3);
y = (s1 *c2* c3) + (c1* s2 *s3);
z = (c1* s2* c3) - (s1 *c2* s3);
```

Virtual sensor

Quaternion to Euler angle conversion



r11=-2*(y*z- w*x) ;

r12=$w^2 - x^2 - y^2 + z^2$**;**

r21= 2*(x*z + w*y);

r31=-2*(x*y - w*z);

r32= $w^2 + x^2 - y^2 - z^2$**;**

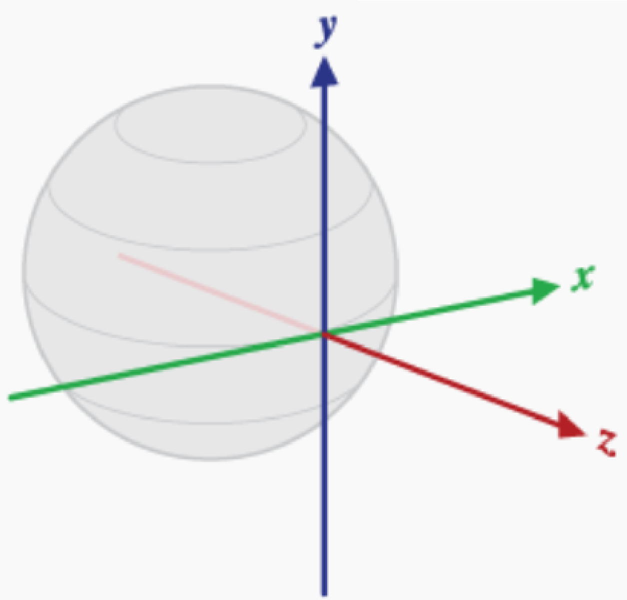
x = atan2(r11, r12);

y = asin(r21);

z = atan2(r31, r32);

Virtual sensor

However w is optional value



values[0]: x*sin(θ/2)

values[1]: y*sin(θ/2)

values[2]: z*sin(θ/2)

values[3]: cos(θ/2) *(optional: only if value.length = 4)*

Virtual sensor

w estimation

$w=(1 - x^2 - y^2 - z^2)$

If $w>0$ then $w=sqrt(w)$

Virtual sensor

How to get orientation from rotation vector

-Quaternion to Euler angle conversion formula(bad)

-Rotation matrix (recommend)

Virtual sensor

How to get orientation from rotation vector

Device flat on a table, top facing north:

1	**0**	**0**
0	**1**	**0**
0	**0**	**1**

Tilted up 30 degrees (rotated about X axis)

1	0	0
0	0.86	-0.5
0	0.5	0.86

Device vertical (rotated about X axis), facing north:

1	0	0
0	0	-1
0	1	0

Device flat on a table, top facing west:

0	-1	0
1	0	0
0	0	1

Virtual sensor

How to get orientation from rotation vector

```
private final float[] mRotationMatrix = new float[9];
private final float[] mRotationMatrix_world = new float[9];
float[] degree=new float[3] ;
```

```
Arrays.fill(mRotationMatrix_world, 0);
mRotationMatrix_world[ 0] = 1;
mRotationMatrix_world[ 4] = 1;
mRotationMatrix_world[ 8] = 1;
```

Point north rotation

```
SensorManager.getRotationMatrixFromVector(mRotationMatrix , event.values);// Create rotation matrix from sensor data
SensorManager.getAngleChange(degree, mRotationMatrix_world, mRotationMatrix);// Calculate angle difference
```

Where degree is store in **[z x y]** in radian format

Virtual sensor

Why we use Quaternion ?

-Handle the problem of the singularities(gimbal lock)
-Easy to interpolate between two quaternion
-Fewer rounding defects
-Faster computation

http://www.phy.ntnu.edu.tw/ntnujava/index.php?topic=2237.0

Thank you ☺