Test de la Géolocalisation sous Android 4.+

Sommaire

I - Capteurs utilisables

I.1 - Capteurs dédiés à la géolocalisation

- GPS : Utilise des données satellite donc technologie d'extérieur.

SDK : Provider : LocationManager.GPS_PROVIDER

Possibilité de choisir la durée minimale entre les updates en ms.

http://developer.android.com/reference/android/location/LocationManager.html#GPS_PROVIDER

- Network : Utilise aussi bien les données du réseau téléphonique que les bornes wifi accessibles. En l'absence de wifi il vaut mieux ne pas etre trop loin de l'extérieur (le fait d'être en intérieur est moins rédhibitoire que pour le GPS)

SDK : Provider LocationManager.NETWORK_PROVIDER

Possibilité de choisir la durée minimal entre les updates en ms.

http://developer.android.com/reference/android/location/LocationManager.html#NETWORK_PROVIDER

Ces capteurs retournent une position géographique (location.Location) qui contient en particulier la latitude, la longitude, l'altitude, la précision de la position et le temps écoulé entre l'allumage du téléphone et l'acquisition des données.

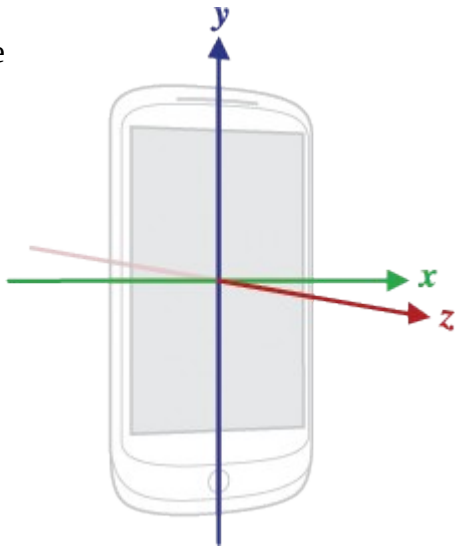
http://developer.android.com/reference/android/location/Location.html

I.2 – Capteurs non dédiés à la géolocalisation

- Accelerometre : Mesure l'acceleration appliquée au téléphone selon x y et z dans un référentiel lié au téléphone. Mesure toutes les forces appliquées au téléphone, y compris la gravité.

SDK : Sensor : Sensor.TYPE_ACCELEROMETER

Quatre vitesses de rafraichissement possibles : Fastest, Game, Normal et UI, par ordre décroissant de rapidité.



http://developer.android.com/reference/android/hardware/Sensor.html#TYPE_ACCELEROMETER

- Gravité : Mesure la l'acceleration appliquée par la gravité au terminal dans le référentiel lié au terminal.

SDK : Sensor.TYPE_GRAVITY

Quatre vitesses de rafraichissement possibles : Fastest, Game, Normal et UI, par ordre décroissant de fréquence

http://developer.android.com/reference/android/hardware/Sensor.html#TYPE_GRAVITY

- Champ magnétique : Mesure le champ magnétique appliqué au téléphone dans le référentiel lié au téléphone.

  SDK : Sensor.TYPE_MAGNETIC_FIELD

  Quatre vitesses de rafraichissement possibles : Fastest, Game, Normal et UI, par ordre décroissant de rapidité.

  http://developer.android.com/reference/android/hardware/Sensor.html#TYPE_MAGNETIC_FIELD

Ces capteurs ne fournissent pas directement de données utilisables mais les données sur les mouvements du téléphone sont utilisables pour déterminer la position du téléphone, la méthode est expliquée en partie II.

I.3 – Capteurs inutiles pour la géolocalisation, contre toute attente

- Gyroscope : Mesure la rotation du téléphone dans le référentiel lié au terminal. Il est inutile dans notre cas car il mesure une vitesse de rotation (en rad/s selon x, y et z) à un instant donné et non l'orientation du téléphone

  SDK : Sensor.TYPE_GYROSCOPE

  Quatre vitesses de rafraichissement possibles : Fastest, Game, Normal et UI, par ordre décroissant de rapidité.

  http://developer.android.com/reference/android/hardware/Sensor.html#TYPE_GYROSCOPE

- Orientation : Mesure l'orientation par rapport au nord magnétique et à la verticale du lieu en rad dans le référentiel lié au terminal. Il est déprécié depuis l'API 8 et remplacé par la méthode SensorManager.getOrientation().

  SDK : Sensor.TYPE_ORIENTATION

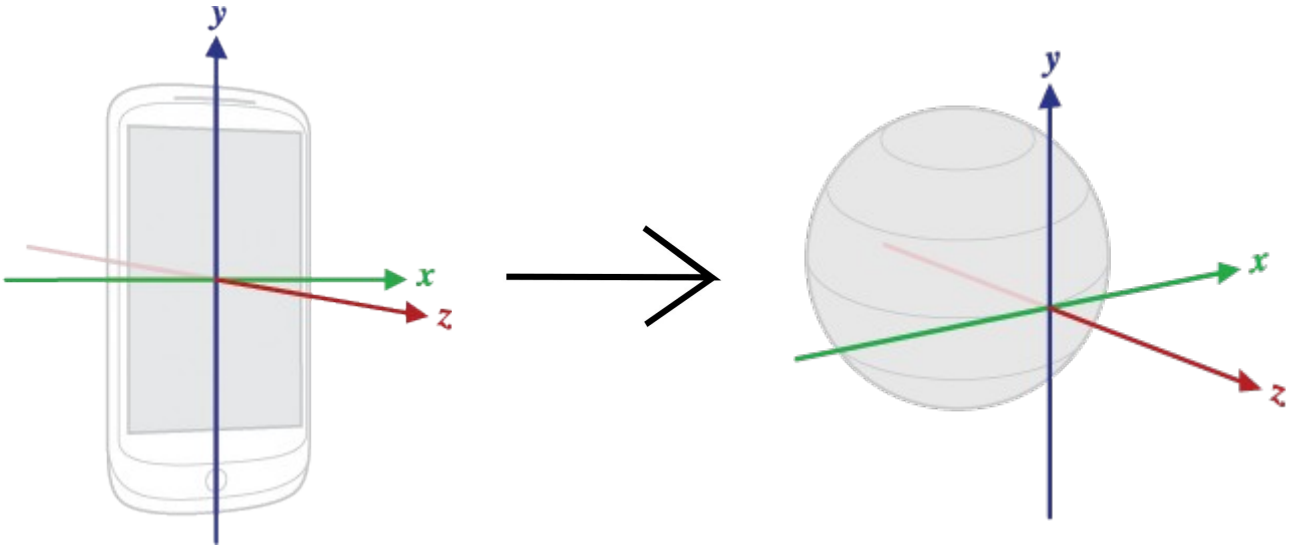
  Quatre vitesses de rafraichissement possibles : Fastest, Game, Normal et UI, par ordre décroissant de rapidité.

  http://developer.android.com/reference/android/hardware/Sensor.html#TYPE_ORIENTATION

II – Calculs permettant d'obtenir la position à l'aide des capteurs de mouvement[1]

II.1 - Obtenir l'accélération du téléphone dans le repère terrestre

La méthode SensorManager.getRotationMatrix(float[] R, float[] I, float[] gravity, float[] geomagnetic)[2], à laquelle on passe des tableaux vides de taille 9 en R et I, l'expression de la gravité renvoyée par le capteur gravity et l'expression du champ magnétique renvoyée par le capteur magnétique. Elle renvoit en R la matrice de rotation qui permet d'exprimer dans un repère terrestre un vecteur exprimé dans le repère du terminal :



Par exemple, prenons l'accélération retournée par l'accéléromètre du téléphone et exprimée dans le repère du téléphone $\begin{matrix} aX \\ aY \\ aZ \end{matrix}$. Si on souhaite l'exprimer dans repère terrestre sous la forme $\begin{matrix} atX \\ atY \\ atZ \end{matrix}$ on va faire un produit matriciel entre l'accélération et la matrice de rotation $\begin{matrix} Rxx & Ryx & Rzx \\ Rxy & Ryy & Rzy \\ Rxz & Ryz & Rzz \end{matrix}$ ainsi :

$$\begin{matrix} atX \\ atY \\ atZ \end{matrix} = \begin{matrix} Rxx & Ryx & Rzx \\ Rxy & Ryy & Rzy \\ Rxz & Ryz & Rzz \end{matrix} * \begin{matrix} aX \\ aY \\ aZ \end{matrix} = \begin{matrix} Rxx.aX + Ryx.aY + Rzx.aZ \\ Rxy.aX + Ryy.aY + Rzy.aZ \\ Rxz.aX + Ryz.aY + Rzz.aZ \end{matrix}$$

Ce qui au niveau du code, si on a l'accélération dans le repère du téléphone dans un tableau float[] acc et que l'on souhaite obtenir l'accélération dans le repère terrestre dans un tableau float[] accT en ayant obtenu avec getRotationMatrix la matrice de rotation float[] R :

accT[0] = acc[0]*R[0] + acc[1]*R[1] + acc[2]*R[2]

accT[1] = acc[0]*R[3] + acc[1]*R[4] + acc[2]*R[5]

accT[2] = acc[0]*R[6] + acc[1]*R[7] + acc[2]*R[9]

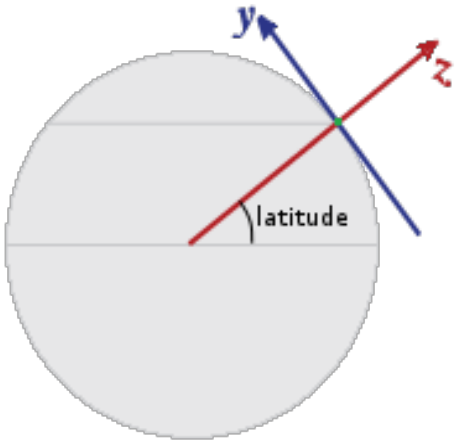
---

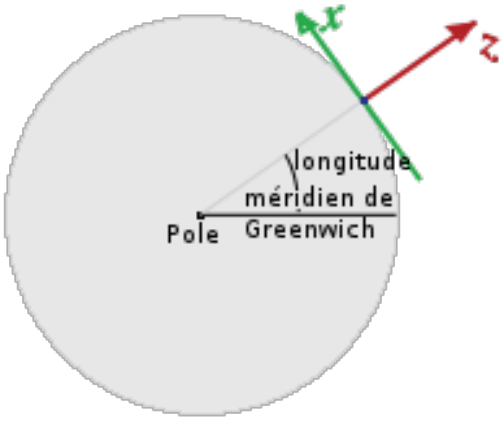
1 http://www.movable-type.co.uk/scripts/latlong.html

2 http://developer.android.com/reference/android/hardware/SensorManager.html#getRotationMatrix(float[], float[], float[], float[])

II.2 – Calculer la vitesse et la direction du terminal à partir de sa position

Pour rappel, la latitude (en °) est l'angle entre la verticale du lieu et le plan de l'équateur et la longitude l'angle entre le méridien où l'on se trouve et celui de Greenwich :





Il ne faut donc pas perdre de vue que les fonctions qu'on applique à ces valeurs doivent fonctionner en °. Si celles ci fonctionnent en radians, Math.toRadians(double angle) permet de faire la conversion des degrés vers les radians et Math.toDegrees(double angle) l'inverse en Java.

On a donc besoin de deux positions (latitude et longitude) consécutives (A et B avec latA, longA, latB et longB) et du temps écoulé entre les deux pour calculer la vitesse moyenne du terminal sur cet intervalle de temps (dt).

Pour cela il faut surtout réussir à calculer la distance parcourue à la surface de la terre pendant cet intervalle de temps. R sera le rayon moyen de la terre. On peut calculer la distance entre les deux points(dist).

dist = R * $\cos^{-1}$ (sin(latA)*sin(latB) + cos(latA)*cos(latB)*cos(longA-longB))

(Il s'agit de la formule de Haversine)

La vitesse (v) peut ensuite être facilement calculée :

v = dist/dt

Quand à l'inclinaison par rapport au nord (i) on l'obtient ainsi :

i = $\tan^{-2}$ (sin(latB-latA)*cos(latB), cos(latA)*sin(latB) - sin(latA)*cos(latB)*cos(longA-longB))

Au niveau du code, la position (pos) renvoyée par le provider (LocationManager.GPS_PROVIDER ou LocationManager.NETWORK_PROVIDER) est donnée sous la forme d'une classe, location.Location, qui contient la latitude (pos.getLatitude()), la longitude (pos.getLongitude()) mais aussi l'instant de l'acquisition de la position (pos.getElapsedRealtimeNanos()) en nano secondes (s * 10^{-9}). Ainsi si on a récupéré avec les providers deux positions consécutives Location posA puis Location posB, on a :

```
//Attention ce code n'est pas testé, il est théorique
double R = 6371000 ; //En mètres
double latA = Math.toRadians(posA.getLatitude()) ; //En radians
```

```
double longA = Math.toRadians(posA.getLongitude()) ; //En radians
double latB = Math.toRadians(posB.getLatitude()) ; //En radians
double longB = Math.toRadians(posB.getLongitude()) ; //En radians
double dt = (posB.getElapsedRealtimeNanos()-posA.getElapsedRealtimeNanos())*Math.pow(10,
-9) ; //En secondes
double dist = R * Math.acos(Math.sin(latA)*Math.sin(latB)  +
        Math.cos(latA)*Math.cos(latB)*cos(longA-longB)) ; //En mètres
double v = dist/dt ; //En m/s
double i = Math.atan2(Math.sin(latB-latA)*Math.cos(latB),
        Math.cos(latA)*Math.sin(latB) – Math.sin(latA)*Math.cos(latB)*Math.cos(longA-longB)) ;
```

II.3 – Obtenir la postion du téléphone en fonction de son déplacement

Pour calculer la position du téléphone en fonction de son déplacement il est nécessaire de connaître sa position initiale et sa vitesse initiale, puisque les capteurs ne nous donnent que l'accélération du téléphone. C'est pour cela que pour utiliser les capteurs pour calculer un déplacement il fallait d'abord utiliser les providers (gps et network) pour connaître sa position (latB et longB) et sa vitesse (v) initiale.

Ensuite, en utilisant l'accélération moyenne exprimée dans le repère terrestre $\begin{matrix} atX \\ atY \\ atZ \end{matrix}$ pour un intervalle de temps donné Δt il est facile de calculer sa vitesse (v_2) et la distance parcourue (d) dans le même repère.

at = $\sqrt{atX^2 + atY^2}$

v_2 = $v + at * \Delta t$

d = v_2 * Δt

Enfin pour calculer la position finale (latF et longF) en utilisant le rayon terrestre (R) et l'inclinaison par rapport au nord calculée précédemment (i) on utilise les formules suivantes :

latF = $\sin^{-1}(\sin(latB)*\cos(d/R)+\cos(latB)*\sin(d/R)*\cos(i))$

longF = $longB+atan2(\sin(i)*\sin(d/R)*\cos(latB),\cos(d/R)-\sin(latB)*\sin(latF))$

Au niveau du code, en considérant que accT[] contient les accélérations moyennes en x, y et z sur un intervalle de temps deltaT :

```
double at = Math.sqrt(Math.pow(accT[0], 2) + Math.pow(accT[1], 2)) ;
double v2 = v + at*deltaT ;
double d = v2*deltaT ;
double latF = Math.asin(Math.sin(latB)*Math.cos(d/R) +
        Math.cos(latB)*Math.sin(d/R)*Math.cos(i)) ;
double longF = longB + Math.atan2(Math.sin(i)*Math.sin(d/R)*Math.cos(latB),
        Math.cos(d/R)-Math.sin(latB)*Math.sin(latF)) ;
```

II.4 – Utilité et limites

Ces calculs peuvent permettre de pallier une insuffisance du GPS et du réseau et les remplacer durant une durée limitée. Tous les calculs tiennent compte de la courbure de la terre ce qui permet théoriquement de garder une bonne précision sur des distances importantes, et la fréquence de rafraichissement des capteurs de mouvement (avec la fréquence au minimum, plus de 60 données par seconde) permet d'obtenir un échantillon de données suffisant.

Cependant il reste un certain nombre d'approximations et de limites :

- Le référentiel terrestre dans lequel est exprimé l'accélération est basé sur le nord magnétique, capteur magnétique oblige. Mais les coordonnées géographiques sont elles calculées par rapport au nord géographique. Cette approximation est plus ou moins génante selon la position du téléphone sur le globe.

- L'utilisation du capteur magnétique pour déterminer les composantes de l'accélération dans le référentiel terrestre rend l'application sensible au champ magnétique ambiant. Ainsi une simple paire d'écouteurs posés contre le téléphone peut fausser les données.

- L'utilisation du capteur de gravité pour déterminer les composantes de l'accélération dans le référentiel terrestre peut s'avérer génant dans un contexte où la gravité ressentie par le téléphone ne serait pas constante : en mer par exemple où le creux des vagues crée des accélérations parasites.

III – Données sur le fonctionnement des capteurs

III.1 Efficacité

Concernant les providers utilisés pour la géolocalisation, il est facile d'obtenir un certain nombre d'informations. Le type de données retourné par ces providers, Location, contient toutes les informations nécessaires.

Pour une position donnée (Location pos) on peut connaître la précision des données en mètres (pos.getAccuracy()) et le temps qui s'est écoulé entre l'allumage du téléphone et l'acquisition des données en nanosecondes (pos.getElapsedRealtimeNanos()).

Pour ce qui est des capteurs (accéléromètre, gravitée, champ magnétique...) il n'y a pas d'information concernant la précision de disponible, cependant les capteurs renvoient quasiment une centaine de données par seconde (en fixant le rafraichissement à sa fréquence minimale, SensorManager.SENSOR_DELAY_UI) ce qui permet si on le souhaite de surveiller la présence de valeurs absurdes avant de faire la moyenne. De même pour connaître la fréquence réelle de rafraichissement, le meilleur moyen reste encore d'utiliser un timer (java.util.Timer) pour calculer à intervalles régulier la fréquence d'acquisition des valeurs. Du fait de la fréquence d'acquisition des valeurs, tout traitement basé sur ces valeurs doit se faire dans un thread séparé ou à intervalles réguliers suffisament espacés pour éviter de surcharger le thread principal (le timer se prête très bien au lancement de traitement dans un thread séparé).

III.2 Consommation de ressources

Lors de l'utilisation d'un sensor (Sensor s) il est possible de connaître sa consommation électrique en mA en utilisant s.getPower().

En revanche pour les providers il n'est pas possible de connaître leur consommation électrique sur un téléphone non rooté.

En fin de compte la manière la plus simple de connaître la consommation électrique d'un capteur est de comparer le niveau de la batterie avant son utilisation. Le code suivant retourne dans float batteryPct une valeur variant entre 0 et 1 (0 pour une batterie vide, 1 pour une batterie pleine) correspondant au taux de charge actuel de la batterie :

```
IntentFilter ifilter = new IntentFilter(Intent.ACTION_BATTERY_CHANGED) ;
Intent batteryStatus = getApplicationContext().registerReceiver(null, ifilter) ;
int level = batteryStatus.getIntExtra(BatteryManager.EXTRA_LEVEL, -1) ;
int scale = batteryStatus.getIntExtra(BatteryManager.EXTRA_SCALE, -1) ;
float batteryPct = level  / (float) scale ;
```

III.3 Informations du téléphone influant sur les valeurs

Si on compare la consommation de plusieurs providers, il est important de prendre en compte un certain nombre de paramètres qui peuvent influer sur le niveau de la batterie au niveau du téléphone.

Parmis ces éléments, les plus importants sont le niveau de luminosité de l'écran, l'activation du wifi, de la connexion au réseau gsm et au réseau de données (3G, H+, 4G, etc).

La luminosité de l'écran est retournée sous la forme d'un entier compris entre 0 et 255 par :

```
android.provider.Settings.System.getInt(getContentResolver(),
        Settings.System.SCREEN_BRIGHTNESS) ;
```

Pour savoir si le wifi est activé le code suivant retourne un booléen (wifiActivated) correspondant à son activation :

```
ConnectivityManager connManager =
        (ConnectivityManager) getSystemService(CONNECTIVITY_SERVICE) ;
NetworkInfo typeWifi = connManager.getNetworkInfo(ConnectivityManager.TYPE_WIFI)
boolean wifiActivated = typeWifi.isAvailable() ;
```

Quand au type de connection au réseau le code suivant permet de savoir si on est connecté au réseau gsm, au réseau de données, au deux ou à aucun :

```
ContentResolver cr = getContentResolver() ;
try{
        int value = Secure.getInt(cr, "preferred_network_mode") ;
        switch (value){
        case 0 :
        case 3 :
                gsm = true ;
                data = true ;
                break ;
        case 1 :
                gsm = true ;
                data = false ;
                break ;
        case 2 :
                gsm = false ;
                data = true ;
        }
} catch (SetingNotFoundException e) {
        gsm = false ;
        data = false ;
}
```

III.4 Informations extérieures influant sur les valeurs

D'autres éléments extérieurs au téléphone peuvent influer sur la consommation de batterie ou sur la précision des valeurs :

- Lorsqu'on se déplace, en particulier dans des zones mal couvertes par le réseau téléphonique, la connection au réseau gsm ou de données consommera plus de batterie que si on reste statique dans

une zone bien couverte.

- La météo peu légèrement influer sur la précision du GPS mais c'est surtout le fait d'être en extérieur ou en intérieur qui va influer sur la durée du fix GPS.

- La température va influer sur la vitesse de déchargement de la batterie (si les températures sont particulièrement basses, la batterie se déchargera plus vite).