Projet: Pedestrian Dead Reckoning for Android

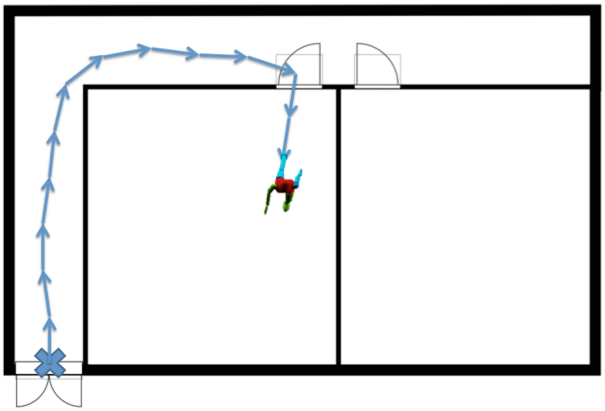
Navigation à l'estime pour piéton pour Android

Thibaud Michel
University of Grenoble Alpes

14 novembre 2016

1 Introduction

Un des problèmes principaux de la localisation sur mobile aujourd'hui est le suivi de la personne là où le signal GPS ne fonctionne pas. C'est notamment le cas à l'intérieur des bâtiments ou encore des zones couvertes (tunnels, grottes...). Nous aimerions par exemple connaître la position d'une personne à l'intérieur d'un musée pour lui proposer du contenu spécifique, ou encore proposer à un mal voyant un système de guidage par localisation. Pour répondre à ce problème, nous avons décidé de ne pas ajouter de nouvelles infrastructures dédiées à la localisation comme le WiFi, les iBeacons ou des émetteurs/récepteurs radio. Ces systèmes là sont utilisés quand la précision est la priorité de l'application. Ce projet s'orientera vers une autre approche qui se nomme Pedestrian Dead Reckoning (ou navigation à l'estime pour piéton en français). C'est une méthode de navigation qui consiste à déduire la position de la personne en fonction de la distance parcourue depuis sa dernière position connue (cf image ci-dessous).



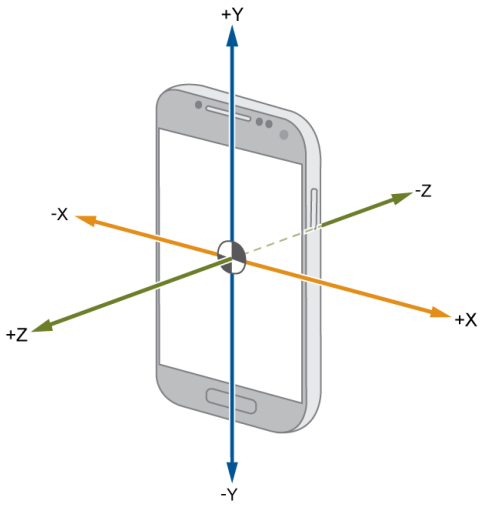
Ce projet est à faire par groupe de 2 personnes, éventuellement 3 mais les attentes seront plus fortes.

Le code de l'application (version finale) devra être transmis aux encadrants avant le xx/xx/2017. Nous porterons beaucoup d'intérêt à l'organisation et la propreté du code. Attention, cela ne se traduit pas par des tonnes de commentaires, mettez-vous simplement à la place d'une personne externe qui doit relire votre code. Autre conseil : Ne surchargez pas votre Activity, elle ne doit pas contenir les éléments techniques de votre application.

Vous devrez aussi faire une présentation de 5-10 minutes le xx/xx/2017 où vous présenterez vos résultats et les difficultés rencontrées. Suivront 5-10 minutes de questions.

2 Podomètre (4h)

Vous avez sûrement déjà utilisé un téléphone, une montre ou encore un autre objet qui compte le nombre de pas que vous réalisez pendant une journée. Tous ces systèmes utilisent au minimum un capteur qui est omniprésent dans les objets connectés : **un accéléromètre**. Ce capteur permet de calculer l'accélération du smartphone suivant les 3 axes (x, y et z) comme défini ci-dessous.



Dans cette première partie vous réaliserez une application qui calcul et affiche le nombre de pas que vous effectuez.

Petits rappels des cours de physique du lycée :

1. Si un objet (à la surface de la terre) est statique alors il sera soumis à l'accélération gravitationnelle qu'on notera g ($\approx 9.8\ m.s^{-2}$).
2. Si cet objet est dans une phase de mouvement, il sera alors soumis à g mais aussi à l'accélération qui provoque son déplacement qu'on appelera **accélération externe**.
3. La gravité est un vecteur qui est toujours dirigé vers le centre de la terre.

Si le téléphone est posé sur une table avec l'écran dirigé vers le ciel, on peut facilement en déduire la gravité, celle-ci sera mesurée sur l'axe z et vaudra environ $9.8\ m.s^{-2}$. Attention, il s'agit bien de $9.8\ m.s^{-2}$ et non $-9.8\ m.s^{-2}$ car on mesure la force exercée par la table sur le smartphone.

Question 2.1 *Utilisez le capteur Sensor.TYPE_ACCELEROMETER d'Android et vérifiez que l'accéléromètre mesure bien* $9.8\ m.s^{-2}$ *sur l'axe* z *et environ* $0\ m.s^{-2}$ *sur les axes* x *et* y *quand il est posé sur la table avec l'écran dirigé vers le ciel.*

Cependant, le smartphone peut être tenu à la main, être rangé dans une poche ou encore un sac à main. Il est alors impossible de savoir si la gravité va être mesurée sur l'axe x, y ou z. On sait seulement que si le smartphone est statique, la norme du vecteur gravité mesurée est égale à $9.8\ m.s^{-2}$. Pour la suite du travail on supposera que l'**accélération** est définie par :

$$\text{accélération} = \|\text{accélération mesurée}\| - \|\text{gravitée}\|$$

Question 2.2 *Vérifiez que l'accélération de votre smartphone est proche de* $0\ m.s^{-2}$ *quand il est statique.*

Certaines études (Thèse de Q. Ladetto) ont montrées que lorsqu'une personne est entrain de marcher, si on regarde l'accélération du centre de masse de cette personne, on peut observer un pic à chaque pas. Ci dessous, un dessin pour expliquer le phénomène :

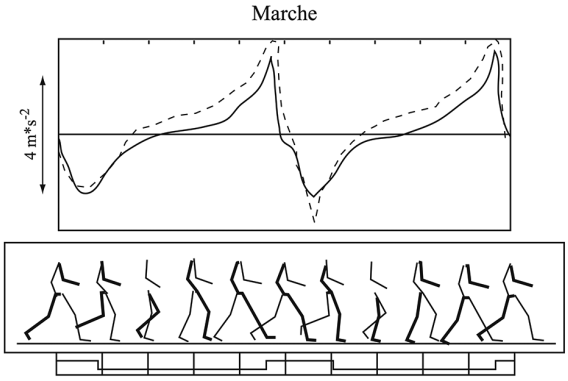


FIGURE 1 – Accélération du centre de masse (courbe pleine) et la même accélération perçue par le smartphone (courbe discontinue) s'il est fixé à la ceinture.

Plus le téléphone sera dissocié du centre de gravité de la personne, moins le podomètre sera fiable. Par exemple il sera plus difficile de compter les pas si le téléphone est dans votre main et que vous effectuez des mouvements de balancier quand vous marchez.

On sait qu'une personne a effectuée un pas si on observe un pic d'accélération au dessus de 3 $m.s^{-2}$. Ce seuil peut varier en fonction des personnes et des mouvements, il pourra alors être modifié si nécessaire.

Question 2.3 *A partir de la norme de l'accélération externe essayez de détecter quand une personne effectue un pas. Aide : Vous pouvez utilisez la détection de front montant.*

Afin d'améliorer cette détection qui est vraiment basique, on va rajouter une condition supplémentaire. Dans ces mêmes études, nous avons aussi remarqué que le temps entre 2 pas n'est jamais inférieur à 0.5 *seconde*.

Question 2.4 *Faites en sorte que votre algorithme prenne en compte la condition précédente.*

Question 2.5 *Implémentez un listener à votre podomètre afin que votre Activity puisse savoir à quel moment un pas a été détecté.*

Question 2.6 *Ajoutez un TextView à votre Activity afin de voir le nombre de pas qui ont été effectués.*

3 Pedestrian Dead Reckoning (4h)

Dans cette partie, nous allons utiliser les différents capteurs du téléphone pour estimer la position de la personne. Le PDR (ou navigation à l'estime) est un système basé sur le positionnement relatif, c'est à dire que la précision de la nouvelle position dépend aussi de la précision de l'ancienne, contrairement au GPS qui fournit des positions (latitude et longitude) qui sont absolues.

La navigation à l'estime pour un piéton est un processus itératif qui prend en paramètres la taille du pas et la direction du pas. Ce processus est enclenché à chaque fois qu'un pas a été detecté :

when a step is detected :

$$\text{position}_{k+1} = calcul\_nouvelle\_position(\text{position}_k,\ \text{taille du pas},\ \text{angle du pas})$$

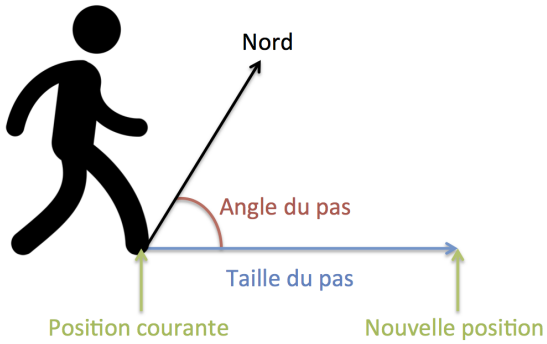


FIGURE 2 – Principe de la navigation à l'estime

Pour la partie suivante nous allons considérer que le smartphone est tenu dans la main avec l'axe y pointant dans la direction de marche et l'écran plutôt orienté vers le ciel.

Dans un soucis de simplicité, nous allons aussi considérer que la taille d'un pas d'une personne est fixe et est de 70cm. Vous pourrez ajuster cette valeur si nécessaire.

3.1 Smartphone Orientation (1h)

La détection de pas a déjà été abordée dans la Section 2, nous allons maintenant essayer de déterminer l'angle du pas effectué. L'angle (α) que nous cherchons est l'angle de la direction de marche relativement au nord. Par exemple, $\alpha = 0°$ si la personne se dirige vers le nord, $\alpha = 90°$ si la personne se dirige vers l'est, $\alpha = 180°$ si la personne se dirige vers le sud ...

La plupart des smartphones que nous avons embarquent un magnétomètre, il est aussi appelé compas ou boussole. Celui-ci nous permet de mesurer le champ magnétique autour du smartphone. La terre, comme la plupart des planètes du Système solaire, possède un champ magnétique. Grâce à ce capteur, le smartphone peut alors estimer la direction du **nord magnétique**. Pour information, le smartphone mesure tous les champs magnétiques. La structure d'un batiment génère parfois de gros champs magnétiques et la direction calculée peut être déviée de quelques degrées.

Nous n'allons pas directement calculer l'orientation à partir des données du magnétomètre. Android embarque un algorithme intelligent qui fusionne les données du magnétomètre, gyroscope et accéléromètre pour nous donner ces informations. On le retrouve dans l'API de Google sous le nom de Rotation Vector.

Question 3.1 *De la même manière dont vous avez utilisé l'accéléromètre dans la partie précédente, utilisez cette fois-ci le capteur Rotation Vector. Le problème est que les valeurs retournées ne sont pas exploitables directement pour ce que nous voulons faire. Je vous demanderais d'utiliser le bout de code suivant :*

```
/**
 * Orientation vector
 * [0] -> Yaw
 * [1] -> Pitch
 * [2] -> Roll
 */
private float[] mOrientationVals = new float[3];

private float[] mRotationMatrixMagnetic = new float[16];
private float[] mRotationMatrixMagneticToTrue = new float[16];
private float[] mRotationMatrix = new float[16];

@Override
public void onSensorChanged(SensorEvent event)
{
    if (event.sensor.getType() != Sensor.TYPE_ROTATION_VECTOR) {
        return;
    }

    // Transforme le rotation vector en matrice de rotation
    SensorManager.getRotationMatrixFromVector(mRotationMatrixMagnetic, event.values);

    // Création de la matrice de passage du repère magnétique au reprère classique
    // http://www.ngdc.noaa.gov/geomag-web/#declination
    Matrix.setRotateM(mRotationMatrixMagneticToTrue, 0, -1.83f, 0, 0, 1);

    // Change la matrice d'orientation du repère magnétique au reprère classique
    Matrix.multiplyMM(mRotationMatrix, 0, mRotationMatrixMagnetic, 0,
            mRotationMatrixMagneticToTrue, 0);

    // Transforme la matrice de rotation en une succession de rotations autour de z, y et x
    SensorManager.getOrientation(mRotationMatrix, mOrientationVals);
}
```

mOrientationVals contiendra 3 angles en radians :

- **Yaw** $[-\pi; -\pi[$ correspond à ce qu'on appelle l'azimuth ou le bearing. C'est la détermination de l'angle que fait, dans le plan horizontal, la ligne du téléphone vers le nord géographique. 0 radian correspond au Nord, $\frac{\pi}{2}$ radian à l'Est... C'est cette valeur que l'on utilisera pour le PDR.
- **Pitch** $[-\frac{\pi}{2}; -\frac{\pi}{2}[$ correspond à la rotation du téléphone autour de l'axe x. On peut par exemple l'utiliser en réalité augmentée pour afficher des objets selon leur hauteur ou altitude.
- **Roll** $[-\pi; -\pi[$ correspond à la rotation du téléphone autour de l'axe y.

Question 3.2 *Vérifier les valeurs de yaw (nord = 0 °, est = 90 °...). N'oubliez pas les conversions radians -> degrés.*

3.2 Calcul de la position (1h)

Maintenant que nous avons la detection de pas, la taille du pas et l'angle du pas il ne nous reste plus qu'à calculer la nouvelle position en fonction de la position courante. C'est à dire la fonction **calcul_nouvelle_position**. Le repère terrestre défini par les coordonnées latitude et longitude n'est pas du tout un repère orthonormé. Nous ne pouvons donc pas utiliser la géométrie 2D classique pour calculer la nouvelle position. Dans notre cas, nous assimilerons la terre à une sphère et vous pourrez vous aider de ce document pour calculer le point de destination en utilisant un arc du cercle.

Question 3.3 *Créez une classe **PDR** avec un attribut **mCurrentLocation** ainsi que les getter et setter associés.*

Question 3.4 *Créez une méthode **computeNextStep(float stepSize, float bearing)** qui prend en entrée la taille et l'angle du pas puis qui met retourne la nouvelle position. Attention aux unités et conversions rad/deg. Dans l'article la latitude et longitude sont en radians et dans votre code, en degrés.*

Question 3.5 *Dans la classe **PDR**, créez 2 méthodes : **start()** et **stop()** qui démarreront et arrêteront la détection de pas et le calcul de l'angle.*

Question 3.6 *Toujours dans la classe **PDR**, abonnez vous à la détection de pas et calculez la nouvelle position quand un pas est detecté*

Malheureusement il ne vous sera pas possible de tester votre implémentation à cette étape car vous n'avez toujours pas de position initiale.

3.3 Cartographie (2h)

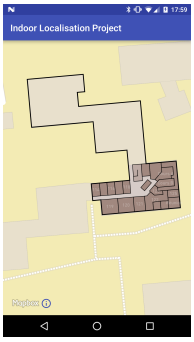
Souvent, pour afficher la position d'une personne sur une carte, on utilise Google Maps et c'est normal car l'utilisation est très facile et il existe beaucoup de tutoriels. Malheureusement l'API est assez limitée quant à la cartographie de l'intérieur des bâtiments. Dans ce projet vous allez utiliser une librairie similaire qui est opensource : Mapbox, à la quelle j'ai rajouté certaines salles du BSHM afin que vous puissiez vous déplacer dans le batiment.

Cette partie est en semi-autonomie, c'est à dire que vous devrez aller chercher sur la documentation de Mapbox certaines des fonctionnalités que vous aurez besoin.

Question 3.7 *Suivez le tutoriel* `https://www.mapbox.com/help/first-steps-android-sdk/` *jusqu'à ce que vous puissiez afficher une carte. Ne restez pas bloqué à cette étape, si vous avez des problèmes n'hésitez pas à demander.*

Question 3.8 *Trouvez la position en latitude et longitude du bâtiment du BSHM. Aide : le site* `https://maps.google.com` *permet de le faire mais ce n'est pas le seul. Une fois trouvé, dans votre Activity, une fois la carte chargée, zoomez sur le bâtiment. Un exemple est donné ci-dessous.*

Question 3.9 *Remplacez le style mapbox par celui-ci :* `http://taha.inrialpes.fr/shs.json`*. Vous devriez voir ceci :*



Question 3.10 *Créez un Marker lorsque vous cliquez sur la carte, si le Marker existe déjà, ne le recréez pas, changez juste sa position.*

Question 3.11 *Branchez le PDR à votre Activity afin de faire déplacer le Marker quand vous marchez. On considerera la position initiale, la position cliquée sur la carte.*

4 Annexes

4.1 Les Capteurs

La plupart des appareils Android ont des capteurs embarqués qui mesurent les mouvements, l'orientation et l'environnement. Ces capteurs fournissent des données brutes avec une bonne précision. Par exemple, dans un jeu on va utiliser le capteur de gravité et l'accéléromètre pour reconnaître des mouvements complexes de l'utilisateur : inclinaison, secousse, rotation ou encore dans une application de voyage l'application utilisera le capteur magnétique et l'accéléromètre pour les utiliser comme boussole. La plateforme Android supporte 3 catégories de capteurs, ils peuvent êtres bruts ou composés (fusion) :

- **Capteurs de Mouvement** : Accéléromètre, Gravité, Gyroscope...
- **Capteurs d'Environnement** : Température extérieure, Pression, Luminosité...
- **Capteurs de Position** : Magnétique, Proximité, Orientation...

On peut accéder à tous ces capteurs grâce au Sensor Framework d'Android. On s'abonne au capteur que l'on veut utiliser grâce à un système de Listener (SensorEventListener). Les données reçues sont au format SensorEvent, on a accès à la précision de la donnée, le capteur duquel elle provient, le timestamp de l'évènement et un vecteur contenant les valeurs en fonction du capteur.

Les capteurs sont souvent très pratiques dans les applications mais il ne faut pas oublier qu'ils sont très gourmands en énergie ! Il est alors important de les utiliser dans les bonnes situations et surtout de s'y abonner seulement quand on en a besoin. Voici un exemple :

```
1  public class SensorActivity extends Activity implements SensorEventListener {
2
3      private SensorManager mSensorManager;
4      private Sensor mLight;
5
6      public void onCreate(Bundle savedInstanceState) {
7          super.onCreate(savedInstanceState);
8          setContentView(R.layout.main);
9
10         mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
11         mLight = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
12     }
13
14     @Override
15     public void onAccuracyChanged(Sensor sensor, int accuracy) { /* Nothing to do */ }
16
17     @Override
18     public void onSensorChanged(SensorEvent event) {
19         // The light sensor returns a single value.
20         // Many sensors return 3 values, one for each axis.
21         float lux = event.values[0];
22         // Do something with this sensor value.
23     }
24
25     @Override
26     protected void onResume() {
27         super.onResume();
28         mSensorManager.registerListener(this, mLight, SensorManager.SENSOR_DELAY_NORMAL);
29     }
30
31     @Override
32     protected void onPause() {
33         super.onPause();
34         mSensorManager.unregisterListener(this);
35     }
36 }
```

4.2 Les Listeners

En programmation évenementielle, il existe plusieurs Design Pattern différents pour recevoir des informations provenants de d'autres entités, les listeners en font parti. Sur Android vous en avez peut être déjà vu avec les boutons d'une interface graphique, lors d'un clic de l'utilisateur un évènement est renvoyé. L'idée est de s'abonner à une entité puis de recevoir une notification à un moment qui n'était pas prévisible. Nous pouvons nous aussi utiliser ce même principe pour surveiller les capteurs (par exemple). Voici ci-dessous un exemple avec un capteur de température.

```
1  public class Thermometer implements SensorEventListener {
2
3          private SensorManager mSensorManager;
4          private Sensor mSensor;
5
6          public Thermometer(SensorManager sensorManager) {
7                  mSensorManager = sensorManager;
8                  mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_TEMPERATURE);
9          }
10
11         public void start() {
12                 mSensorManager.registerListener(this, mSensor,
13                         SensorManager.SENSOR_DELAY_NORMAL);
14         }
15
16         public void stop() {
17                 mSensorManager.unregisterListener(this);
18         }
19
20         @Override
21         public final void onSensorChanged(SensorEvent event) {
22
23                 float temperature = event.values[0];
24
25                 if(temperature > 30 && mTemperatureListener != null) {
26                         mTemperatureListener.onHightTemperatureDetected(event.values[0]);
27                 }
28         }
29
30
31
32         private TemperatureListener mTemperatureListener;
33
34         public void setTemperatureListener(TemperatureListener listener) {
35                 mTemperatureListener = listener;
36         }
37
38         public interface TemperatureListener {
39                 public void onHightTemperatureDetected(float temperature);
40         }
41 }
```

Dès que le capteur de température dépasse 30°, l'objet abonné reçoit alors une notification grâce à la méthode **onHightTemperatureDetected()**

```
public class TemperatureActivity extends Activity {

        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main);

                SensorManager sensorManager =
                        (SensorManager) getSystemService(Context.SENSOR_SERVICE);

                Thermometer mThermometer = new Thermometer(sensorManager);
                mThermometer.setListener(mTemperatureListener);
        }

        @Override
        public void onResume() {
                super.onResume();
                mThermometer.start();
        }

        @Override
        public void onPause() {
                super.onPause();
                mThermometer.stop();
        }

        private TemperatureListener mTemperatureListener = new TemperatureListener() {

                public void onHightTemperatureDetected(float temperature) {
                        Toast.makeText(getApplicationContext(),
                                "High temperature detected: " + temperature,
                                Toast.LENGTH_SHORT).show();
                }
        };
}
```