Travail

Ewan BURASOVITCH - BURE30090400

Dans le cadre du cours :
Département d’informatique et mathématiques

Application mobile

Date : 25 Mars 2025

Session hiver 2025

Université du Québec à Chicoutimi

Rapport sur deux capteurs Android

Travaux Dirigés du 18 et 25 mars

---

1. Choix des capteurs

Voici les deux capteurs que j'ai sélectionné de la liste proposée dans le TD :

- **Capteur de pression atmosphérique**
- **Compteur de pas**

---

2. Fiche technique des capteurs

Capteur de pression atmosphérique

Type et constante Android :

- Type de capteur : Capteur environnemental (baromètre).
- Constante : Sensor.TYPE_PRESSURE

Unité de mesure et format des données :

- L'unité est l'hectopascal (hPa) ou millibar (1 hPa = 1 mbar).
- Le format de données renvoyé est un tableau float[], mais la valeur principale est values[0], qui correspond à la pression en hPa.

Précision et fréquence d'échantillonnage :

- Précision : de l'ordre de ±1 hPa, selon les constructeurs.
- Fréquence : configurable (p. ex. SENSOR_DELAY_NORMAL, SENSOR_DELAY_UI, etc.). Généralement, on ne nécessite pas d'échantillonnage très élevé (quelques fois par seconde suffisent).

Disponibilité sur différentes gammes d'appareils :

- La présence d'un baromètre n'est pas systématique. De nombreux smartphones milieu/haut de gamme en sont équipés (Samsung Galaxy, certains Google Pixel, etc.), mais d'autres gammes (entrée de gamme) peuvent en être dépourvues. Le taux de présence peut être estimé entre 40 % et 60 % selon les générations.

Permissions nécessaires :

- Aucune permission dangereuse n'est requise pour accéder à ce capteur. L'accès est libre via l'API standard d'Android.

Consommation énergétique :

- Relativement faible. Le baromètre ne génère pas de consommation élevée, surtout à faible fréquence d'échantillonnage.

Variations chez les fabricants et dépendances matérielles :

- Selon le composant utilisé (Bosch, STMicroelectronics, etc.), on peut observer de légères différences de précision et d'étalonnage.
- Certains appareils compensent la mesure en fonction de la température interne pour améliorer la précision.

Compteur de pas

Type et constante Android :

- Type de capteur : Capteur de position/mouvement spécifique.
- Constante : Sensor.TYPE_STEP_COUNTER

Unité de mesure et format des données :

- Valeur : Nombre de pas (cumulatif).

- Format de données : values[0] renvoie un float correspondant au total de pas détectés depuis que le capteur a été initialisé (ou depuis le dernier redémarrage de l'appareil).

Précision et fréquence d'échantillonnage :

- Précision : En général bonne, mais des erreurs de comptage peuvent survenir (faux positifs ou pas non détectés).
- Fréquence : Actualisation à chaque nouveau pas détecté. Il ne s'agit donc pas d'un signal continu, mais d'un incrément événementiel.

Disponibilité sur différentes gammes d'appareils :

- Présent sur la plupart des appareils récents (souvent dans le SoC ou le coprocesseur de mouvement). Sur certains anciens modèles, il n'existe que l'accéléromètre, et la détection de pas doit être faite manuellement par l'application.
- Taux de présence en croissance, car les constructeurs intègrent de plus en plus ce capteur dédié.

Permissions nécessaires :

- Généralement, aucune autorisation "dangereuse" n'est requise pour lire ce capteur. Sur certaines versions d'Android, l'utilisation de l'API Activité (ou d'autres capteurs liés à la santé) peut potentiellement nécessiter des permissions, mais pour le Sensor.TYPE_STEP_COUNTER, c'est souvent libre d'accès.

Consommation énergétique :

- Très basse, car la détection de pas est souvent gérée par un coprocesseur de mouvement à faible consommation (et non par un traitement permanent sur l'accéléromètre principal).

Variations chez les fabricants et dépendances matérielles :

- Selon le matériel, l'algorithme de détection de pas peut varier et entraîner des comptages différents.
- Certains constructeurs affinent les algorithmes pour réduire les faux positifs (détections de secousses comme pas).

---

3. Cas d’utilisation

Capteur de pression atmosphérique

Application météo locale : Estimer l’altitude et la tendance (haussière ou baissière) de la pression pour prévoir des changements climatiques.

Détection de changement d’altitude : Par exemple, dans une application de randonnée, estimer le dénivelé pour fournir des informations sur les performances ou l’itinéraire.

Contrôle d’une station météo : Coupler la pression à d’autres capteurs (température, humidité) pour un tableau de bord environnemental.

Compteur de pas

Suivi sportif : Calculer le nombre de pas quotidiens dans une application de fitness.

Déclenchement d’alertes : Lancer automatiquement une notification ou un rappel hydratation si l’utilisateur a dépassé un certain nombre de pas.

Gamification : Créer un jeu où chaque pas compte comme un point virtuel pour encourager l’utilisateur à se déplacer.

---

4. Mini-exemples de code

Extrait de code pour le capteur de pression atmosphérique

```
private SensorManager sensorManager;
private Sensor pressureSensor;
private SensorEventListener pressureListener;

@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
   pressureSensor = sensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE);

   pressureListener = new SensorEventListener() {
      @Override
      public void onSensorChanged(SensorEvent event) {
         float currentPressure = event.values[0]; // en hPa
```

```
            // Traitement : mise à jour de l'interface, enregistrement en base de données, etc.
        }

        @Override
        public void onAccuracyChanged(Sensor sensor, int accuracy) {
            // Gérer la variation de précision si nécessaire
        }
    };
}

@Override
protected void onResume() {
    super.onResume();
    if (pressureSensor != null) {
        sensorManager.registerListener(pressureListener, pressureSensor,
                SensorManager.SENSOR_DELAY_NORMAL);
    }
}

@Override
protected void onPause() {
    super.onPause();
    if (pressureSensor != null) {
        sensorManager.unregisterListener(pressureListener);
    }
}
```

Extrait de code pour le compteur de pas

```
private SensorManager sensorManager;
private Sensor stepCounterSensor;
private SensorEventListener stepCounterListener;

private float initialStepCount = 0;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    stepCounterSensor = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER);

    stepCounterListener = new SensorEventListener() {
        @Override
```

```
        public void onSensorChanged(SensorEvent event) {
            float totalSteps = event.values[0];

            // Pour calculer le nombre de pas depuis le lancement de l'app:
            if (initialStepCount == 0) {
                initialStepCount = totalSteps;
            }
            float stepsSinceAppStart = totalSteps - initialStepCount;
            // Mettre à jour l'UI, envoyer des données, etc.
        }

        @Override
        public void onAccuracyChanged(Sensor sensor, int accuracy) {
            // Gestion de la précision si besoin
        }
    };
}

@Override
protected void onResume() {
    super.onResume();
    if (stepCounterSensor != null) {
        sensorManager.registerListener(stepCounterListener, stepCounterSensor,
                SensorManager.SENSOR_DELAY_NORMAL);
    }
}

@Override
protected void onPause() {
    super.onPause();
    if (stepCounterSensor != null) {
        sensorManager.unregisterListener(stepCounterListener);
    }
}
```

Intégration avec WorkManager

Pour automatiser la récupération et l'envoi de données (ex. upload sur un serveur toutes les heures), on peut utiliser WorkManager :

```
public class SensorUploadWorker extends Worker {

    public SensorUploadWorker(@NonNull Context context,
                              @NonNull WorkerParameters workerParams) {
```

```
        super(context, workerParams);
    }

    @NonNull
    @Override
    public Result doWork() {
        return Result.success();
    }
}
```

Ensuite, on planifie la tâche :

```
PeriodicWorkRequest uploadWorkRequest =
    new PeriodicWorkRequest.Builder(SensorUploadWorker.class, 1, TimeUnit.HOURS)
        .build();

WorkManager.getInstance(context).enqueue(uploadWorkRequest);
```

---

5. Évolution du capteur et différences entre versions

Évolution du baromètre (pression atmosphérique)

- **Versions Android** : Présent depuis plusieurs années. Les premières implémentations n'offraient pas forcément de modes de "batching".
- **Améliorations récentes** : Android a introduit des "wake-up sensors" vs. "non wake-up sensors" (à partir d'Android 5.0), permettant de mieux gérer la consommation.
- **Fabricants** : Certains fabricants ajustent la pression en tenant compte de la température interne ou de l'humidité pour plus de précision.

Évolution du compteur de pas

- **Introduit dans Android 4.4 (KitKat)** via l'API TYPE_STEP_COUNTER et TYPE_STEP_DETECTOR.
- **Améliorations** : Les algorithmes se sont perfectionnés, notamment chez certains constructeurs (Samsung, Google Pixel) qui disposent de coprocesseurs dédiés.
- **Fabricants** : Les différences se situent surtout dans la fiabilité du comptage. Certains modèles enregistrent plus de faux pas que d'autres.

---

6. Intégration dans un projet de tâches contextuelles

Exemple d'utilisation :

Projet de suivi d'activités quotidiennes :

- Le **baromètre** peut servir à estimer le dénivelé lors d'une marche ou d'une montée d'escaliers.
- Le **compteur de pas** fournit directement le nombre de pas effectués.

Algorithme / heuristique :

Si l'utilisateur dépasse un certain nombre de pas par jour *et* qu'une différence de pression significative est détectée (indice de montée), l'application peut déclencher un scénario (p. ex. l'envoi d'un message de félicitations ou la mise à jour d'un score de gamification).

---

7. Observation pratique

Pour l'observation :

Installation d'une application d'analyse (ex. *Sensor Kinetics*, *Physics Toolbox Sensor Suite*).

Vérification de la disponibilité :

Sur mon appareil, j'ai observé la présence d'un capteur de pression (Sensor.TYPE_PRESSURE) et d'un compteur de pas (Sensor.TYPE_STEP_COUNTER).

Mesures relevées :

- **Le baromètre** affichait environ 1012 hPa au repos, fluctuant légèrement quand je changeais d'altitude (monter/descendre un étage).
- **Le compteur de pas** incrémentait son total à chaque fois que je faisais un pas (testé sur ~20 pas).

Deux idées d’applications innovantes :

- **Application de navigation intelligente** : Avec la pression atmosphérique, estimer l’étage où se trouve l’utilisateur dans un grand immeuble. Le compteur de pas aide à déterminer la progression dans les couloirs/escaliers.

- **Challenge de marche en altitude** : Combiner la variation de pression (altitude approximative) avec le nombre de pas pour proposer des défis sportifs (atteindre un certain “pic” virtuel).

Références

- Documentation Android sur les capteurs

- API SensorManager

- *Sensor Kinetics* et *Physics Toolbox Sensor Suite* (applications tierces pour l’observation)

- *Android Developers Blog* – articles sur l’optimisation de la consommation et la gestion des capteurs