Vol.2013-DBS-157 No.2
Vol.2013-IFAT-111 No.2
2013/7/22

センサーネットワークデータベースでの計測データモデルの検討とＧＰＳセンサーでの検証

藤井達也[†1]　　金子邦彦[†2]

現在、Android 端末のような安価な端末で、GPS レシーバ、３軸加速度、磁気などのデータが計測できるようになってきた。これらセンサーを用いた種々の応用が行なわれるようになってきた。一方で、複数の Android 端末をセンサーと見立て、サーバマシンに複数種類のセンサーデータを収集するとき、そのデータ型を記述する計測データモデルが課題となる。本稿では、センサーでの計測データがレコード形式であると仮定し、レコードの属性名と値のペアを、連想配列の一要素に見たてる。これで、複数種類のセンサーの計測データが連想配列の形式で記述できるようになる。Android 端末の複数種類からのデータ収集で、その記述法を評価する。

A Study on a Measurement Data Model for Sensor Network Database and its Evaluation using GPS Sensor Data

TATSUYA FUJII[†1]　KUNIHIKO KANEKO[†2]

Recently, Android terminals have many types of sensors, such as GPS, G-sensor, E-compass, etc. There are many applications using their sensor. A problem when collecting multiple types of sensors from multiple numbers of Android terminals is a description model to describe data type of sensor data. In this report, we assume a record format for sensor datasets, and each record is associatve array of pairs of record attribute name and attribute value. Multiple types of sensor data can be described using such description model. We evaluate the description model by some experimental tests.

1. はじめに

Android 端末を使用すれば特別なセンサーが無くても加速度、磁気、GPS のデータを計測できる。複数の Android 端末から複数種類のセンサーデータを収集するとき、センサーデータのデータ形式が問題になる。本稿では、JSON ファイル形式を使うことで、複数の Android 機器からの複数種類のセンサーデータの収集と処理が容易にできることを実験によって示す。

2. Android 端末

表 1　本稿で使用する Android 端末

	Google Nexus 7	SONY NW-Z1050
Android OS	4.2	2.3
ディスプレイサイズ	7 インチ	4.3 インチ
3 軸加速度センサー	あり	あり
磁気センサー	あり	あり
GPS レシーバ	あり	あり

Android 端末には種々のセンサーが搭載されている。本稿で使用する Android 端末は Google Nexus7 と SONY NW-Z1050 の 2 機種であり、センサーは以下の 3 種である。

- 3 軸加速度センサー
- 磁気センサー
- GPS レシーバ

各機種の概要を表 1 に示す。なお、上記 3 種類のセンサーは、今回使用した Android 機器の療法に搭載されている。センサーで取得できる情報を表 2 に示す。

表 2　センサーで取得できる情報

センサー	属性名	データ型	精度	単位
3 軸加速度	x	float	32bit	m/s^2
	y			
	z			
磁気	x	float	32bit	μT
	y			
	z			
GPS レシーバ	Latitude	double	64bit	
	Longitude			

2.1 Android プログラム開発環境

Android 機器のソフトウエア開発には Google が配布している Android ソフトウエア開発キット（Android SDK）を

†1 九州大学
Kyushu University
†2 九州大学
Kyushu University

使用する。Android SDK に付属するツールとして、デバッグモニタ、Android Debug Bridge (adb)があり、プログラムの開発に便利であった。Android ソフトウエアでは、Android SDK が提供する Java 言語ライブラリを使用する。本稿でも、適宜、使用した Java 言語ライブラリのクラス名、メソッド名などを記す。

3. 測定ソフトウェア

3.1 使用した主な API

3.1.1 ID を取得するための API

下記のコードで、シリアル ID が取得できる。

```
String serialId = android.os.Build.SERIAL;
```

端末固有の識別番号は IMEI や IMSI など複数存在するが、携帯電話ではない端末でも問題なく取得できたのがシリアル ID だったので、これを使用している。

3.1.2 測定を開始するための API

以下、本稿では、適宜、クラスのインスタンス名を「m＜クラス名＞」のように記す。センサーでの測定を開始するためのコードを表 3 に示す。このコードは onResume アクティビティ（ソフトウェアがアクティブになったときに実行されるアクティビティ）に書く。

「アクティビティ名」には、センサーを取得したいアクティビティ名を書く。そのアクティビティには特定のクラスが implements されている必要がある。必要なクラスを表 4 に示した。

表 3　測定を始めるためのコード

センサー	コード
3 軸加速度	mSensorManager.registerListener (アクティビティ名, mSensorManager.getDefaultSensor (Sensor.TYPE_ACCELEROMETER), SensorManager.SENSOR_DELAY_FASTEST)
磁気	mSensorManager.registerListener (アクティビティ名, mSensorManager.getDefaultSensor (Sensor.TYPE_ MAGNETIC_FIELD), SensorManager.SENSOR_DELAY_FASTEST)
GPS レシーバ	mLocationManager.requestLocationUpdates (LocationManager.GPS_PROVIDER, 0,0, アクティビティ名);

表 4　必要なクラス

センサー	クラス
3 軸加速度	SensorEventListener
磁気	SensorEventListener
GPS レシーバ	LocationListener

3.1.3 センサーからデータを取得するための API

センサーの値を取得するためのコードを表 5 に示す。また、全てのコードは、センサーの値が変わった時に自動で呼び出される関数内に書かなければならない。表 6 にセンサーと関数名の対応を示した。

表 5　センサーから測定値を取得するときのコード

センサー	属性名	コード
3 軸加速度	x	mSensorEvent.values[0]
	y	mSensorEvent.values[1]
	z	mSensorEvent.values[2]
磁気	x	mSensorEvent.values[0]
	y	mSensorEvent.values[1]
	z	mSensorEvent.values[2]
GPS レシーバ	Latitude	mLocation.getLatitude()
	Longitude	mLocation.getLongitude()

表 6　センサーの値が変わった時に呼び出される関数

センサー	関数名
3 軸加速度	onSensorChanged(SensorEvent)
磁気	onSensorChanged(SensorEvent)
GPS レシーバ	onLocationChanged(Location)

3.2 ソフトウェアの概要

3 軸加速度、磁気、GPS レシーバからデータを取得し、全てのデータを 1 つの JSON ファイルに保存する。

測定の開始時点で機体のシリアル ID を保存し、全ての測定データにタイムスタンプを付けて保存することによって、機体のシリアル ID と時間を指定すれば、その時点の 3 軸加速度、磁気、GPS レシーバの測位情報のデータが取り出せるようにした。サンプリングレートは、プログラムの書き換えによって 4 段階で設定することができるが、同じ設定でも機種や負荷によって変化する。3 軸加速度センサー、磁気センサーからは 3 軸の計測値 (x, y, z) が取り出せる。それらは図 1 に示す Android 実機の局所座標系での値である。

Vol.2013-DBS-157 No.2
Vol.2013-IFAT-111 No.2
2013/7/22

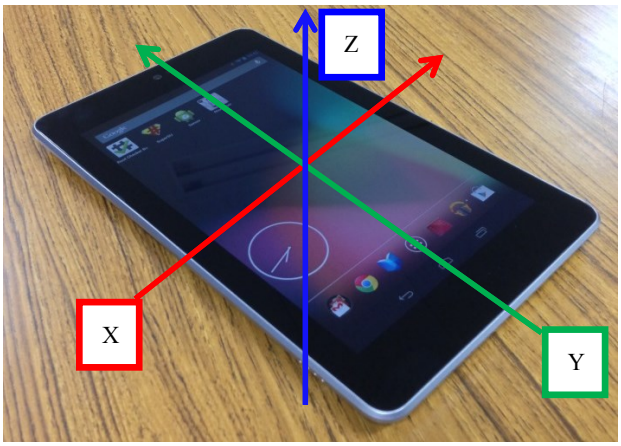

図 1　Android 実機に局所な座標系

3.3 プログラムの手順

1. 機体のシリアル ID を調べて JSON オブジェクトに追加する。これは下記のコードになる。

```
String serialId = android.os.Build.SERIAL;
```

2. タイマーをセットし、JSON オブジェクトを一定の時間間隔でファイルに出力し続けるよう設定する。100ms 間隔で出力するように設定した。

3. センサーの値が更新された時に実行されるアクティビティの中で、jsonObject.put(キー, 値);を実行し、JSON オブジェクトに測定値を追加する。

3.4 JSON ファイル

3.4.1 測定の区切りについて

測定開始時に"{"を出力し、測定終了時に"}"を出力するようにしてあるので、一連の測定は 1 つの JSON オブジェクトの中に入る。一度測定を止めてから次の測定を始めるともう一つの JSON オブジェクトが作られる。

3.4.2 キーについて

キーになり得る文字列には 4 つの種類がある。それらのキーと値との関係を表 7 に示した。タイムスタンプは、System.currentTimeMillis();で取得できるデータをそのまま使っている。これは 1970 年 1 月 1 日から経過したミリ秒数である。

表 7　キーと値の関係

キー	値
ID	機体のシリアル ID
タイムスタンプ,a	加速度センサーの値
タイムスタンプ,m	磁気センサーの値
タイムスタンプ,g	GPS で取得した緯度,経度

3.4.3 センサーの種別について

キーのタイムスタンプの後に付いている文字で、センサーの種類を識別できるようにする。図 2～4 に JSON ファイルの例を示した。

```
{{"ID":"015d490215040218"}
{"1371476063174,m":"x,-13.324637,y,8.426633,z,-24.492537","137147
{"1371476063331,m":"x,-14.148242,y,10.350161,z,-23.362871","13714
{"1371476063485,a":"x,0.8539818,y,5.932832,z,8.0684595","13714760
{"1371476063696,a":"x,-1.1547534,y,7.101202,z,6.743269","13714760
{"1371476063767,a":"x,-0.61486095,y,6.9144545,z,6.95755","1371476
{"1371476063910,m":"x,-12.759373,y,8.438949,z,-23.986456","137147
```

図 2　Google Nexus7 で全て測定したファイル(先頭部分)

```
{"ID":"015d490215040218"}
{"1370415691448,a":"x:0.35748425,y:0.36197338,z:9.887305","137041
{"1370415691560,a":"x:0.34072483,y:0.35359368,z:9.876531","137041
{"1370415691656,a":"x:0.39818567,y:0.32725745,z:9.896882","137041
{"1370415691689,a":"x:0.35748425,y:0.35239655,z:9.904064","137041
{"1370415691841,a":"x:0.38022915,y:0.32246906,z:9.93878","1370415
{"1370415691970,a":"x:0.35269585,y:0.35598788,z:9.910049","137041
```

図 3　Nexus7 で加速度のみ測定したファイル（先頭部分）

```
{{"ID":"8036559"}
{"1371591600588,m":"x,14.86718,y,11.526765,z,-23.044199","1371591
{"1371591600765,a":"x,1.0573393,y,-0.6901286,z,9.291448","1371591
{"1371591601009,a":"x,-1.105822,y,-2.0871453,z,7.3377795","137159
{"1371591601096,m":"x,13.425869,y,5.1986456,z,-26.241735","137159
{"1371591601246,a":"x,5.114314,y,-6.8480153,z,11.573122","1371591
{"1371591601298,a":"x,5.4650645,y,-6.994062,z,13.083863","1371591
```

図 4　SONY NW-Z1050 で全て測定したファイル（先頭部分）

3.4.4 サンプリングレート

同じ設定で、加速度のみの測定と加速度と磁気と GPS の同時測定を行ったところ、サンプリングレートに違いが出た。サンプリングレートを表 8 にまとめた。

表 8　Google Nexus7 のサンプリングレート

	加速度	磁気	GPS
3 軸加速度のみ測定	191 回/秒		
3 軸加速度、磁気、GPS レシーバーを同時に測定	144 回/秒	88 回/秒	0.16 回/秒

表 9　SONY NW-Z1050 のサンプリングレート

	加速度	磁気	GPS
3 軸加速度のみ測定	52 回/秒		
3 軸加速度、磁気、GPS レシーバーを同時に測定	48 回/秒	12 回/秒	

Google Nexus7 での測定において GPS レシーバは 1 秒ごとに測定するが、取得できないことが多いのでサンプリン

グレートが低くなっている。

本稿での実験では、SONY NW-Z1050 での GPS データ取得は、以下の理由により断念した。SONY NW-Z1050 での測定では GPS の電波をキャッチできない場合がある。GPS レシーバは搭載されているが、我々の作成したプログラムでは、データ測定中にパソコンに繋いでいないとデータを取り出せないという予想外の現象があり解決できていない。パソコン付近では、GPS の信号が安定的に受信できないという現象に出会ったため、GPS データ取得は断念した。解決は今後の課題である。

3.4.5 ファイルサイズ

加速度と磁気と GPS の同時測定を Google Nexus 7 で行った時のファイルサイズ増大の速さは、加速度部分が 7.88K バイト/秒、磁気部分が 4.82K バイト/秒、GPS 部分が 12.16 バイト/秒であった。

SONY NW-Z1050 で行った場合は、加速度部分が 2.63K バイト/秒、磁気部分が 0.65K バイト/秒であった。

4. センサーでの測定

4.1 測定環境

表 10 の測定環境において、加速度の測定を行った。

1. Android 機器に Google Nexus7 を使用した。
2. サンプリングレートを最速に設定した。
3. ズボンの左前ポケットに Android を上下逆向きに入れた。
4. 259 秒間、測定した。測定の内訳は表 10 の通りである。

表 10　測定の内訳

時間	動作
0～10.2 秒	測定開始の作業
10.2～38.9 秒	階段上り
38.9～88.8 秒	歩行
88.8～110.1 秒	階段上り
110.1～206.8 秒	歩行
206.8～222.6 秒	エレベーターを待機
222.6～244.4 秒	エレベーター(3 階から 8 階)
244.4～259 秒	測定終了の作業

4.2 プロットのための準備

R のプロット用ライブラリ(scatterplot3d)でデータを扱えるようするため、JSON 形式から CSV 形式へ変換し、タイムスタンプとセンサーの種類でソートした。

変換プログラムは Ruby で作成した。JSON ファイルを解析し、キー→値→改行コードの順に出力する。プロットには、ID は必要ないので除去している。プログラム名：convert.rb , JSON ファイル名 : 1.json , CSV ファイル名 : 1.csv としてプログラムファイルを保存し、次のシェルプログラムにより変換とソートを行う。

```
ruby convert.rb 1.json | fgrep -v m | sort > 1.csv
```

図 5 に先頭部分を示した JSON ファイルを CSV ファイルに変換したものを図 5 に示す。

```
1371591600573,a,  x,2.992752E-4,y,-0.95708215,z,10.2000475
1371591600574,a,  x,-0.009277532,y,-0.99658644,z,9.554811
1371591600580,a,  x,0.1678934,y,-0.96905315,z,9.305813
1371591600584,a,  x,0.24211365,y,-0.9403227,z,9.383625
1371591600585,a,  x,0.42287588,y,-0.87448215,z,9.921123
1371591600586,a,  x,0.48273093,y,-1.1390414,z,9.702053
```

図 5　変換後の CSV ファイル（先頭部分）

4.3 3 次元プロット

集まったデータを、scatterplot3d を使用して散布図にした。

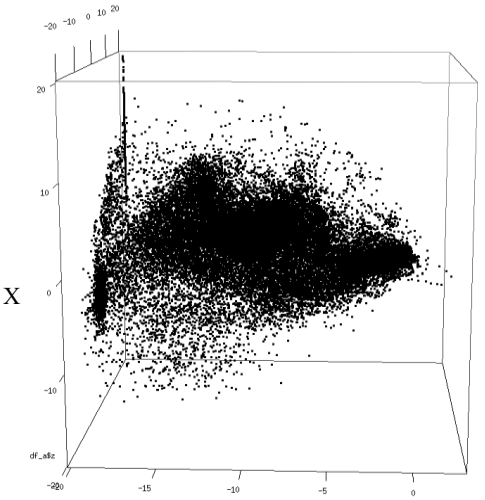

図 6　加速度の 3 次元散布図

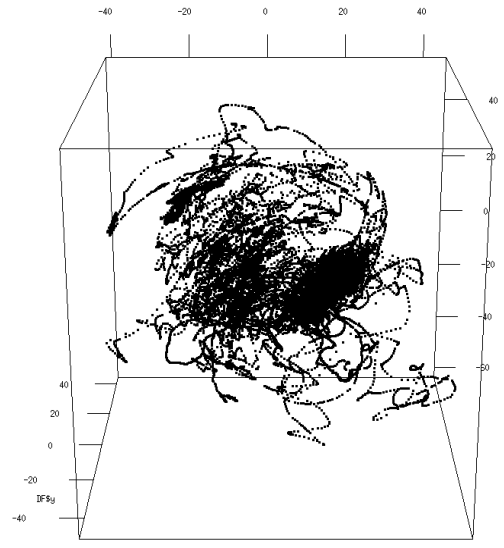

図 7　磁気の 3 次元散布図

Vol.2013-DBS-157 No.2
Vol.2013-IFAT-111 No.2
2013/7/22

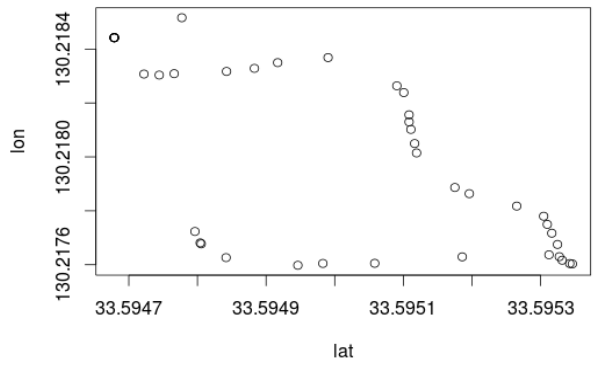

図 8　GPS データの 2 次元散布図

- 加速度の数値の単位はm/s^2である。
- 磁気の数値の単位はμTである。
- 目盛は自動で決められる。
- 3 次元散布図は、マウスホイールでズームイン/ズームアウトできる。
- 3 次元散布図は、マウスをドラッグすると、散布図を回転させることが出来る。

4.4 加速度の大きさをプロット

3 次元プロットだけでは測定データの時間変化が分かりづらいので、縦軸(Y)を$\sqrt{x^2+y^2+z^2}$として、横軸は時間として 2 次元のプロットを行った。

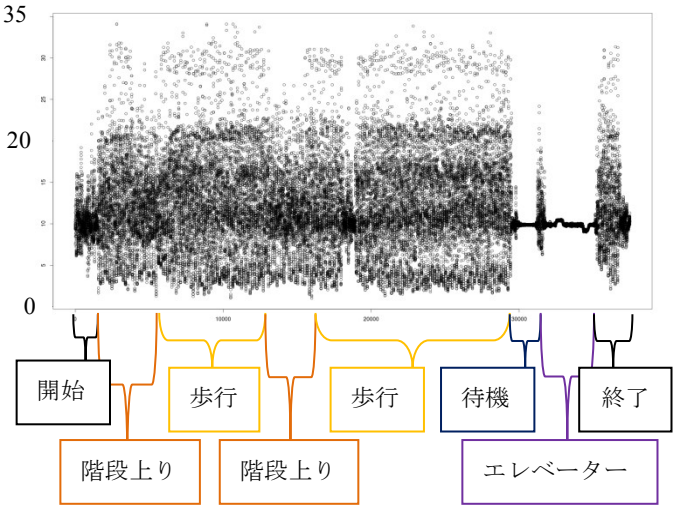

図 9　加速度の大きさの時間変化

歩行時の Y=20 付近に、密度が高い部分があるのが分かる。センサーの測定限界が±20 m/s^2だから、1 つの軸が測定限界でその他の軸が 0 に近い状況において Y=20 程度となるためだ。ここで改めて 3 次元の散布図を見てみると、y 軸の加速度が頻繁に-20m/s^2になっていることが分かる。これは足をついたときに、強い上向きの加速度が現れているからだと予想される。

タイムスタンプでソートした CSV ファイルを見てみると、測定の間隔が一定ではなく、大きなばらつきがあることが容易に確認できる。図 10 に例を示す。

```
1371591600592,m,  x,14.198216,y,11.582199,z,-23.113384
1371591600593,m,  x,14.074675,y,11.483499,z,-23.30244
1371591600594,m,  x,13.959972,y,11.383351,z,-23.480135
1371591600721,m,  x,13.863951,y,11.285841,z,-23.633724
1371591600722,m,  x,13.738093,y,11.046497,z,-23.923143
1371591600723,m,  x,13.623145,y,10.841997,z,-24.137573
```

図 10　測定間隔のばらつきの例

測定間隔が広すぎる部分を無くすのは難しいが、測定間隔が狭すぎる部分を無くせばデータサイズを減らせるはずである。

5. おわりに

データ計測の結果、データファイルができ、各センサーのサンプリングレートや、サンプリング間隔のばらつきなどが容易に観測できる。種類の異なるセンサーであっても、JSON ファイルで計測でき、散布図等のグラフを容易に作成できる。以上のことを確認できた。

謝辞

本研究は（独） 新エネルギー・産業技術総合開発機構, IT 融合による新社会システムの開発・実証プロジェクト（テーマ名：移動体データ銀行で実現する次世代交通情報共通基盤アジアモデルの構築）の助成を受けたものです。