東海大学大学院2014年度 修士論文

節電の取り組みを可視化するAndroidアプリの実装と評価

Design and Implementation of Android App Visualizing Power Saving

指導教員 菊池浩明, 石井啓之 教授

東海大学大学院 情報通信学研究科 情報通信学専攻

2BJNM004 大久保成晃

目 次

第1章　はじめに

東日本大震災による原発の事故により電力供給量の低下とそれに伴う脱原発の動きがあり，節電の必要性が高まってきている．企業や官公庁，学校など多くの組織が様々な方法で節電に取り組んでいる．例えば，冷房の設定温度を上げたり，クールビズ，サマータイムの導入，エレベータの稼動数を減らす「上下 3 階以上の移動には階段を使いましょう」といった取り組みやパナソニックの節電取り組み [1] や，[2] などがある．更に，個人での取り組みにまで要請されている．例えば，スマート家電による自宅の家電の消し忘れ防止や，ELP Lite[3] という身の回りの家電の電力消費量を可視化するハードウェアなどである．Nest Learning Thermostat[4] では，部屋の温度を自動で調節し，必要ないときは自動で判断し電気を節約する．

しかし，個人での節電を支援する取り組みは，会社や公共施設での取り組みと比べてその努力が目に見えにくい．ひとりでエレベータの利用を控えて階段を進んで使ったとしても，取り組みが形に残らず，節電に貢献している実感が薄い．家庭では，節電の取り組みは最終的に電気代の請求書という形で本人に返ってきて，どれだけ節電しているか実感がわくのに対して，階段の利用は証拠が残らない．

そこで，本研究では公共の場における節電のみえる化を行い，個人に対しどの程度の貢献ができたのかを可視化することでこの問題の解決を試みる．日常の運動量から個人の健康状態を可視化，向上することで，家庭と同様に利益が発生する仕組みを考える．個人が階段を使っているのか，エレベータを使っているのか，という状態を，Android 端末に掲載されるセンサを用いて，利用者の状態を自動推定する Android アプリケーション NNCloud を開発する．

本研究の新規性は次のとおりである．(1) ユーザの四状態（歩行，階段，エレベータ，停止）を自動判定するアルゴリズムを考案する，(2) ユーザが階段を使って節電した量を電力に換算して表示する，(3) 提案システムの実現可能性を示すため，5 日間の運用実験を行う．本実験結果に基づき，状態推定の精度と本システムで見積もられる電力量の信頼性を評価する．

第2章　提案システム(NNCloud)

2.1　NNCloudの概要と目的

NNCloud[10]は，ユーザがエレベータを使う代わりに階段を使った時間を測定し，節電の効果を確認可能にするシステムである．階段を歩行していることをAndroidに搭載されるセンサで自動推定することにより，利用者がエレベータを使う頻度を知覚させて，その利用を抑え，節電を推奨する．逆にエレベータを使えば節電力量が減っていくことで，エレベータの利用を控えるように利用者にうながす．普段は記録にも残らないエレベータの利用を控えたことによる節電量を，明確な数値としてユーザに提示することで意欲の向上を図る．

2.2 NNCloud の仕様

本節では NNCloud で必要となる各種パラメータについて述べると共に，その仕様について述べる

2.2.1 階段歩行による節電量の見積もり

NNCloud でもっとも重要な節電量を見積もる．エレベータを用いずに階段歩行による節電力量を次の様に見積もった．

まず，一般的な 15 人乗りエレベータ [5] の仕様より，消費電力を 12 kw とする．このエレベータを 1 階から 6 階まで昇降するのに必要な時間は，学内で計測したところ平均 20 秒であった．したがって，エレベータによる消費電力量 E は，

$$E = 12000\frac{20}{3600} = 66.6[\mathrm{W}]$$

である．これを，階段を使って 1 階から 6 階まで昇降したと仮定する．実測による経過時間は 90 秒であった．よって，以上の値から，階段を昇降するのに 1 秒当たり $E/90 = 0.74$ [W] 節電できる．逆にエレベータを利用する場合，1 秒当たり 3.33 [W] 消費する．

2.2.2 行動推定

行動は停止，歩行，階段，エレベータに分けられる．この行動を推定する必要がある．行動推定はセンサの値を考慮して 5 秒間のサイクルで行う．推定アリゴリズムは次のとおりである．

1. 歩数アルゴリズムを実行し，歩行の推定を行う．
2. 歩行していた場合は，次に階段の推定を行う．階段の条件を満たした時「階段」と出力，そうでない場合は「歩行」と出力する．
3. 歩行していない場合，エレベータの推定を行う．エレベータだった場合は「エレベータ」と出力，そうでない場合は「停止」と出力する．

(1) 歩数アルゴリズム

歩行と判定するアルゴリズムは以下のとおりである．図 2.1 は，停止状態から 40 秒後，階段を昇り始め，その後 110 秒頃から歩行した時の加速度センサ値 dy と Roll[度] の変化を表している．加速度が 10 [m/s^2] を中心に増減しているのがわかる．まず，加速度センサから得られる加速度 dy と，地磁気センサから得た値を使った端末の傾きを組み合わせ，地面と垂直な加速度を導き出す．その後，逐次入力された垂直加速度

を最大値が更新されるたびに調べ，重力 $g = 9.8m/s^2$ を下回った際，今度は最小値を調べる．次に，重力 g を上回った際に最大値と最小値の差が閾値 0.98 を超えたか否かで「歩行」と判別する．

(2) 歩数の推定

歩いているかどうかは歩数アルゴリズムで求めた歩数が閾値 θ を超えているかどうかで判定する．本方式では 5 秒間のセンサデータを推定する．そのため閾値は一秒に一歩と考え，データの欠損を考慮し $\theta = 4$ 歩と設定した．

(3) 階段の推定

歩行と推定された場合，それが階段を歩いているのか，水平な道を歩いているのか切り分ける必要がある．本方式では，歩行と階段を切り分けるために Android 端末の傾きを表す Pitch と Roll の値に注目した．図 2.1 の加速度は歩行時も階段の昇降時も大きく変化している．しかし，Roll は階段を昇降している時に変動が小さかったのに対し歩行時は大きく変動しているのが確認できる．図 2.2 は歩行時と階段の昇降時の Pitch のヒストグラムである．図 2.2 では, 階段の昇降が 100 度に偏っているのに対し，歩行はばらつきが大きい．これらのことから，Pitch と Roll の分散の合計 σ^2 が閾値 $\theta_\sigma = 800$ を下回った際に「階段」であると推定する．

(4) エレベータの推定

図 2.3 にエレベータに乗車時の Z 軸方向の加速度の変化を示す．発進時と停止時に大きな加速度の変化が確認できる．この特徴を微分し，それらの最大値と最小値から閾値 80% 以内の値を探す．見つけた値と対応する逆の特徴，正の値を見つけた場合は負の値を，負の値を見つけたときは正の値を探し，それぞれ特徴が閾値 1.9s 以内だった場合エレベータと推定する．また，エレベータは発進時と停止時以外の時は特徴が現れないため，エレベータが発進した特徴を推定したのち停止状態だった場合もエレベータと推定する．

2.3 実装

提案システムのクライアントを Java 言語を用いて，Android 上に実装した．バージョン 2.2 から 4.2 までに対応している．

端末はどのような向きで携帯されるか分からない為，常に地面から垂直方向の加速度の変化を読み取るように X,Y,Z 方向のセンサの値を正規化する必要がある．これには，加速度センサと地場センサを用いた [1)]．

状態を記録する為に，大量のセンサデータを逐次格納する必要がある．この目的の為に，データベースに格納し，必要に応じて取り出すことを考えた．しかし，データベースはセンサデータを書き込み続ける中取り出さなくてはならず，必然的に I/O が足りなく処理が遅いことが分かった．そのため ArrayList を利用して格納し，終わったらオブジェクトを削除する方式にした．

図 2.4 は作成した NNCloud の実行例である．現在の状態を表すフィールド（上）とその一日の行動を振り返ることを可能にした履歴のフィールド（下）の二つから構成されている．図の例では，現在状態は「停止」で，その日の累積歩行数 544 歩であることを示している．履歴フィールドの操作は，上下フリックでスクロールするとその日一日をさかのぼり，左右フリックで過去の日付への移動ができる．図の例では，5 月 14 日に 2,555 歩の歩行したことを表している．

サーバとの通信は POST/HTTP を用いる．サーバは HTTP サーバで php で作成した API により，ユーザの登録，ログイン，マイルの送信，ログの送信を提供する．ユーザ登録はランダムで一意な疑似 ID とパスワードを発行することによって行われる．ユーザはサーバとの通信を意識することなく利用することができる．

サーバとの通信は 1 時間に 1 度行う．もし接続に失敗した場合は次のタイミングで接続時に再送信する．

[1)] 従来広く用いられていた ORIENTATIN センサは，現在では失効された deprecated method に分類されている．ORIENTATION センサは実際には加速度センサと地場センサを複合して実装されているので，今後のメンテナンスを考え加速度センサと地場センサの値から直接端末の傾きを計算している

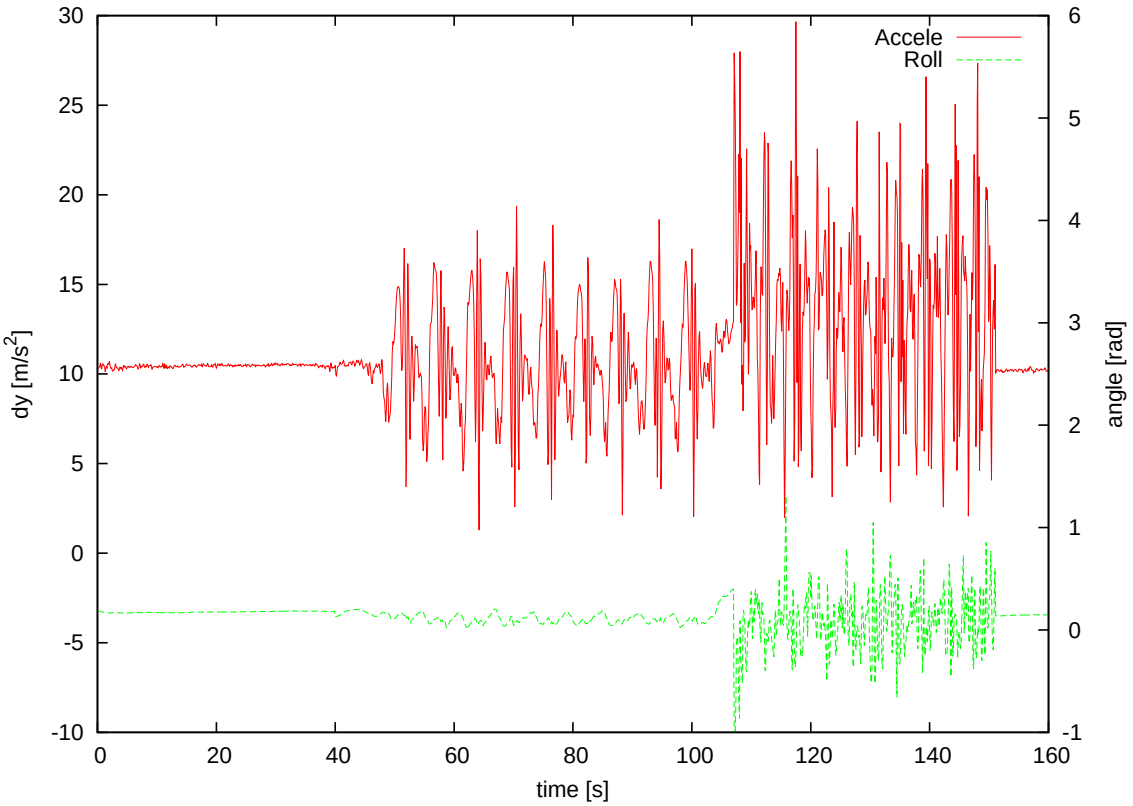

図 2.1: Z 軸加速度センサ dy と Roll の変化

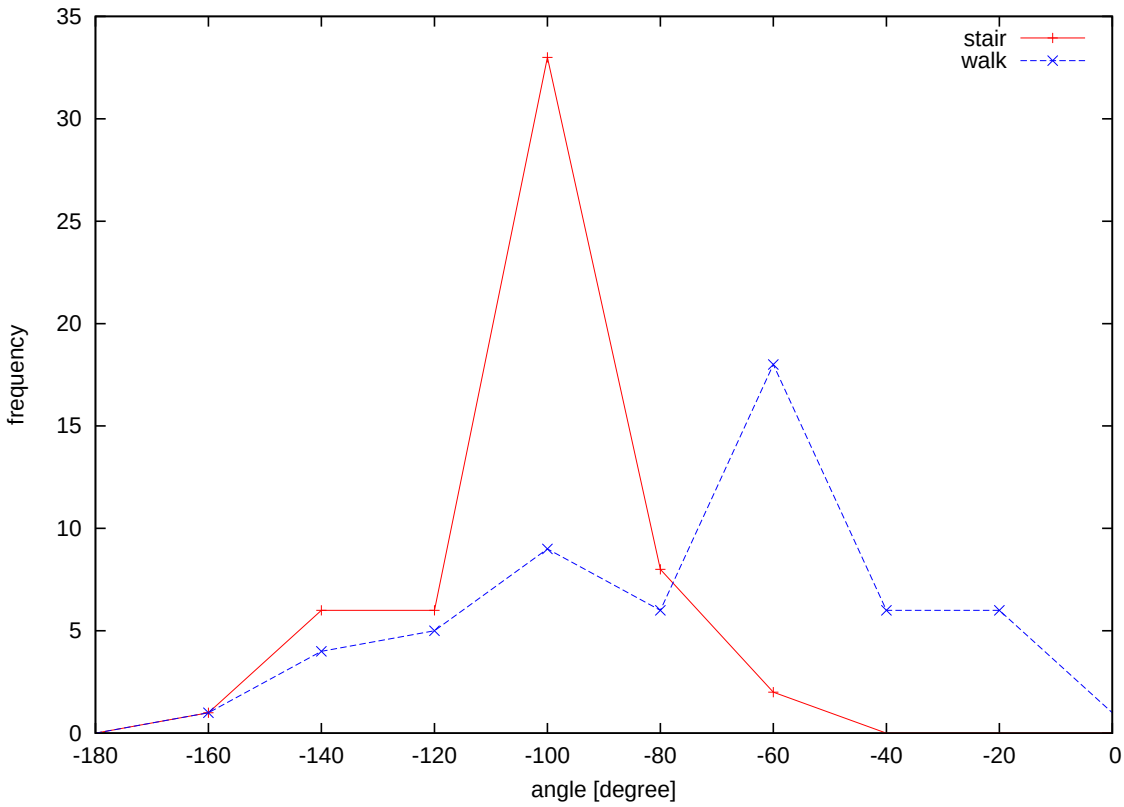

図 2.2: 階段と歩行時の Pitch ヒストグラム

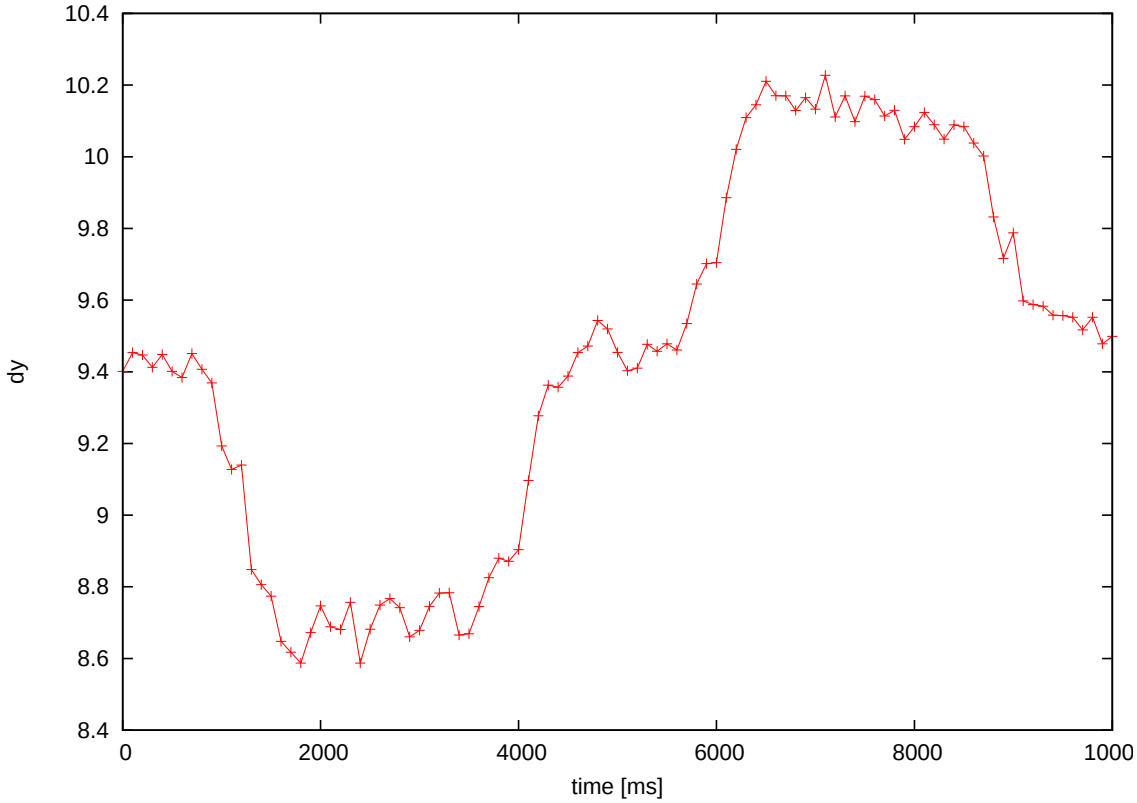

図 2.3: エレベータ内の Y 軸加速度の変化

図 2.4: NNCloud の実行例

第3章　評価

ここでは，実装したシステムを実際に運用しその有効性を示す.

3.1 実験 1: 状態推定精度

提案方式の精度を次の方法で評価した．2013 年 3 月に，5 名の被験者に，端末を持った状態で大学構内で，歩行，階段の昇降，エレベータの乗車を順に行ってもらい，センサの値を測定した．エレベータの一往復の時間が 40 秒であったため，被験者は各状態を 40 秒測定することとした．

判定結果を表 3.1 に示す，入力データの単位は秒である．平均適合率は 0.58 である．

表 3.1: 提案方式の精度 [s]

真 ＼ 判定	walk	stair	elevator	stop	sum
walk	165	35	0	0	200
stair	115	80	0	5	200
elevator	15	25	110	50	200
stop	0	15	80	105	200
適合率 p	0.83	0.4	0.55	0.53	
平均 $\overline{p}$	0.58				

3.2 実験 2: 運用実験

長期間での使用に問題なくユーザの動作が測定できるか，どこまで節電できるかを調べるため以下の実験を行った．7 人の被験者を募り 2013 年 5 月 8 日から 14 日まで運用実験を行った．被験者は大学生と教員である．

実験 1 によりエレベータが停止時に誤推定される割合が高いことが分かったため，エレベータを推定せずに実験を行うこととした．歩数のアルゴリズムの誤差を補うために,「万歩計」Android アプリの Walkroid[7] を利用した．被験者には終了時 Walkroid の歩数と NNCloud の歩数それぞれを申告してもらった．

図 3.1 に，ユーザ u_7 の歩行量の時間分布を表す．ウィークディである 8, 9, 10 日に関しては，通学時の 11 時頃と帰宅時の 19 時前後に大きなピークが見られる．

図 3.2 は，全被験者の二つのアプリ NNCloud と Walkroid の測定値の散布図である．Walkroid が多くの場合 NNCloud より高い歩数を測定しているが，逆のケースもいくつか見られる．アルゴリズムの精度には個人差があることが示されている．

表 3.2 に，NNCloud で推定した状態の統計量を示す．一日のうち，9 割以上が「停止」であり，活動している期間は短い．

表 3.2: 判定状態の割合（7 名）[s]

	平均	標準偏差	割合
stop	29398	20451	0.9814
walk	262.04	470.66	0.0087
stair	295.10	561.42	0.0099

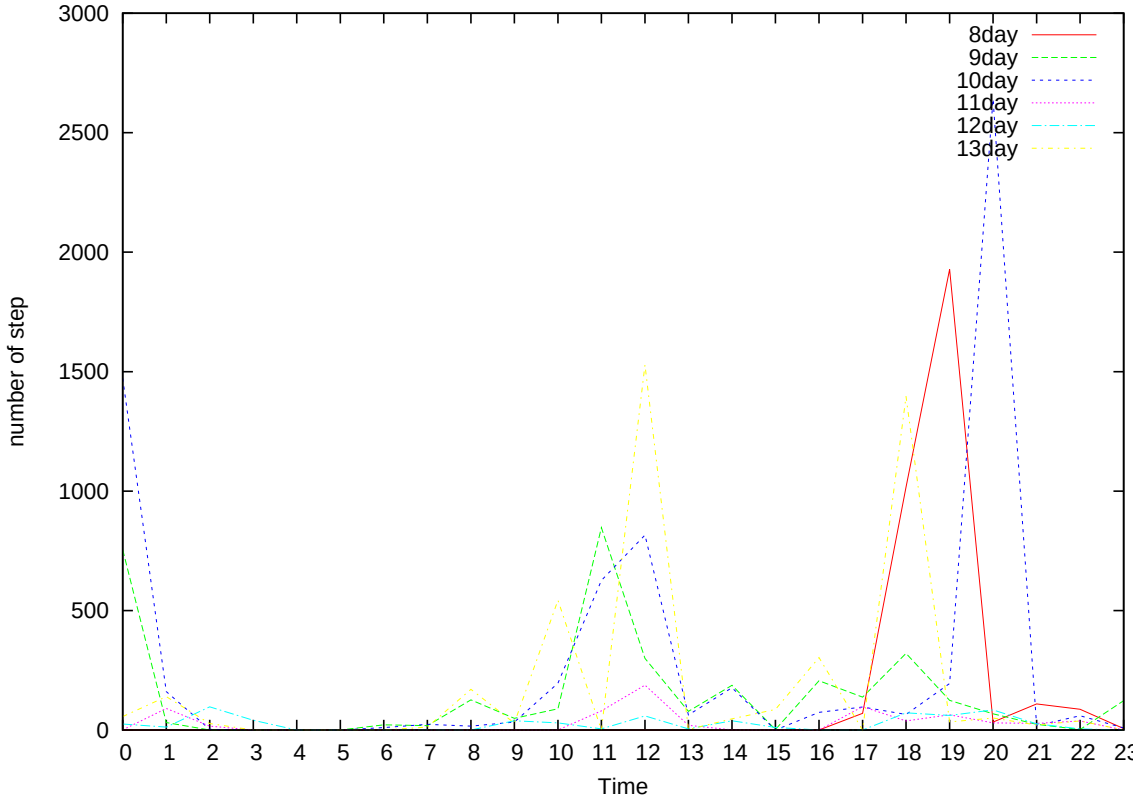

図 3.1: 歩数の時間分布（u_7）

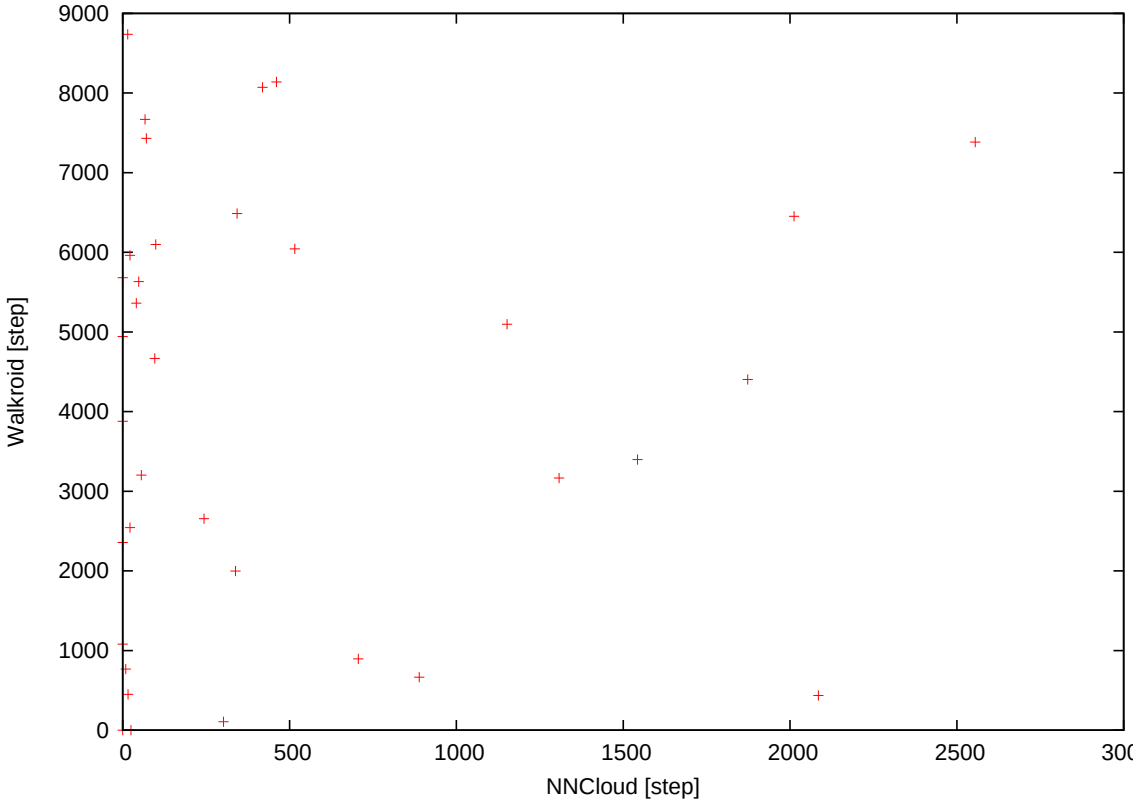

図 3.2: NNCloud と万歩計アプリ Walkroid の歩数散布図

3.3 節電効果

階段の利用量を電力に換算して全ユーザの総節電量を調べる，図 3.3 では，利用者が 10 時ごろ出かけたのち，20 時ごろに帰宅した際に利用していた階段の昇降時間とそれによる節電力量の関係を表している．ユーザが階段を使うことで節電量が増加していることが確認できる．

図 3.4 は，本実験で全ユーザの節電の節電力量である．ユーザ u_1 が最も多く節電量を記録している．ユーザ間でこのような節電量を比較することで，節電の意欲が向上することを期待している．

以上の結果より，本実験では，7 名 1 週間で 15,160 [W] 節電できたことが明らかになった．

3.4 考察

実験 1 の表 3.1 より，停止時または階段昇降時にエレベータと誤推定してしまうことが多いことが分かった．階段昇降時に誤推定されるのは，垂直加速度の計算に誤りがあり，端末の向きによっては重力加速度がマイナスになってしまうためである．停止時の誤推定は，端末の微弱な振動を特徴量としてとらえてしまい，エレベータと誤推定してしまっていたことが原因である．

実験 2 の図 3.4 では被験者 u_1 が大きく節電量を記録している．これは u_1 が意識して特に歩き回っていたことと，端末の持ち方が一定で歩行アルゴリズムが正しく動作できていたためである．Android 端末の持ち方が悪かったり，鞄に入れて持ち歩いたり，個人差が大きいことが分かった．それぞれで歩行の判定しきい値を学習するような工夫が今後必要である．

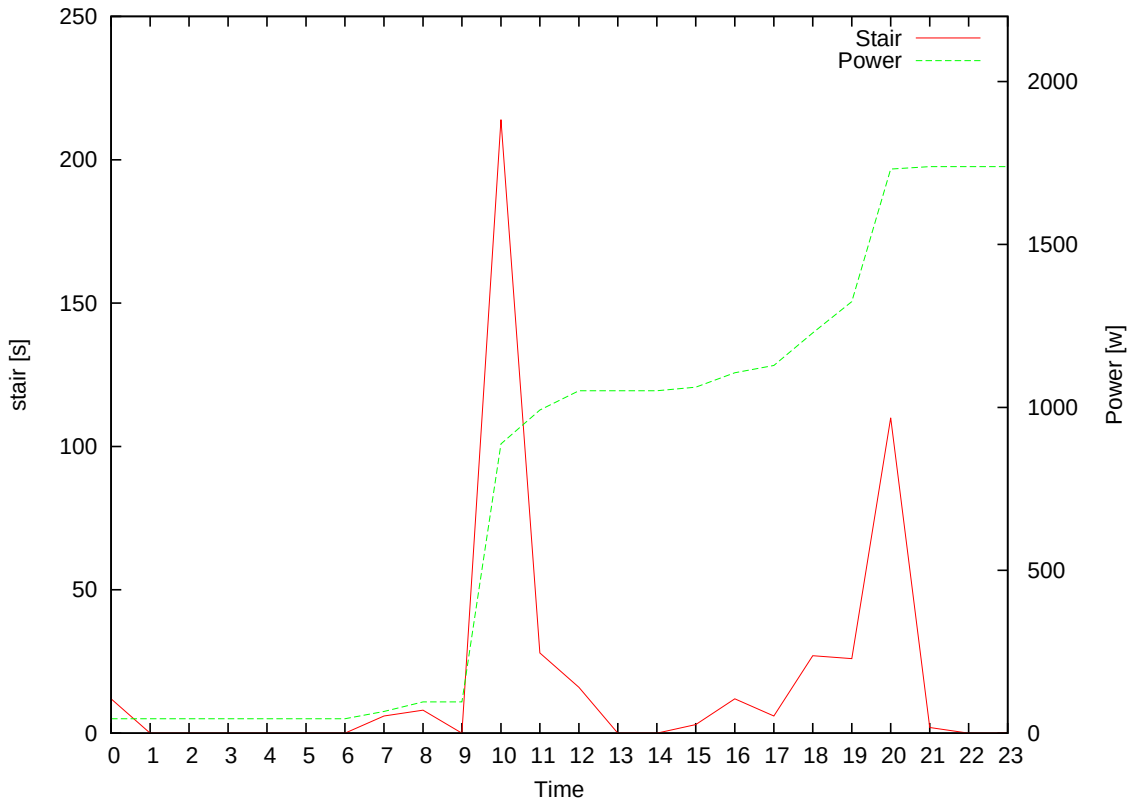

図 3.3: 階段歩行時間と累積節電力量

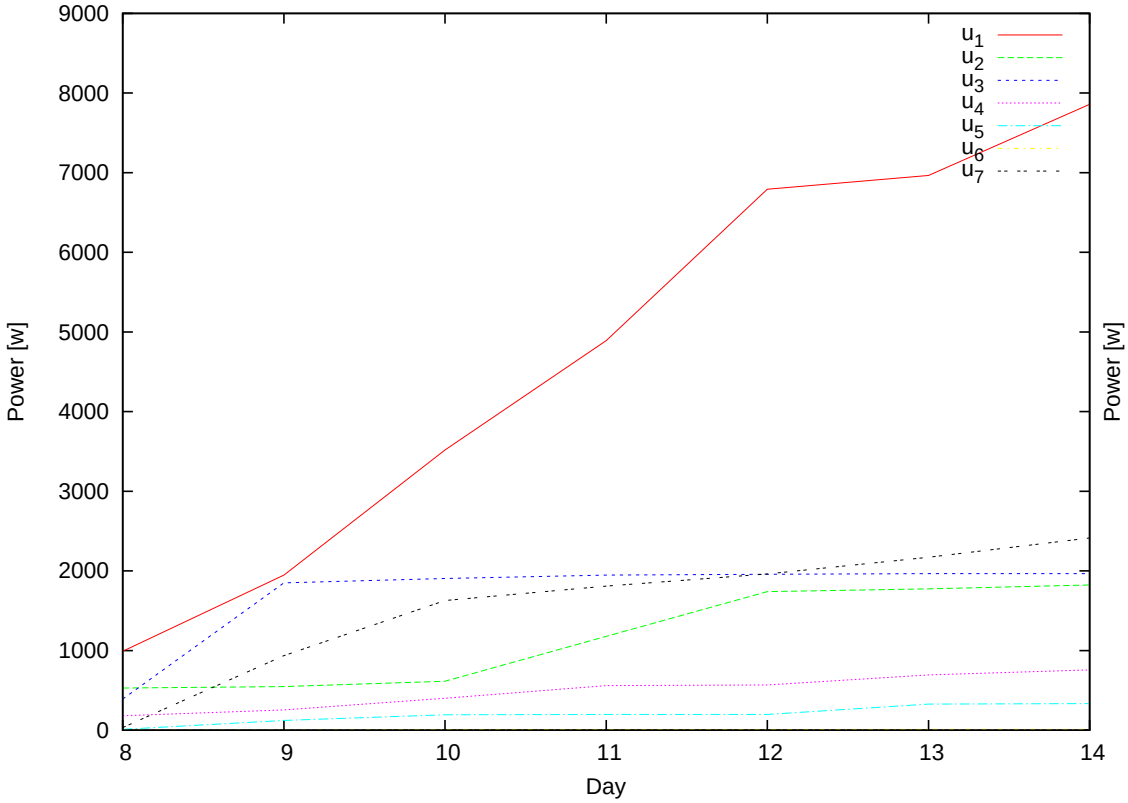

図 3.4: ユーザ毎の累積節電力量

第4章　おわりに

本論文は節電に対する意識が高まっている中，具体的に節電効果を目に見えるかするシステムとしてNNCloudを提案した．NNCloudは利用者の状態として停止，歩行，階段，エレベータのいずれかを自動推定し，階段を歩いて節電した量を自動計測するものであり．その基本機能の確認と運用実験の結果を行った．その結果，初期の目的を果たした．

今後はNNCloudのアルゴリズム改善を行い，ユーザ数を増やした長期的な運用実験を行う．また，歩数アルゴリズムは垂直加速度の計算式の修正に加えて，微弱な加速度のスキップを行うエレベータ判定方式の改良が必要である．更に，推定状態を増やし，エスカレータ，自転車，自動車などをも節電の対象として広げていきたい．ポータルサイトを作成しユーザの行動や，ほかのユーザの節電力量などを確認することも計画している．

参 考 文 献

[1] パナソニックの節電取り組み (http://panasonic.co.jp/eco/setsuden/) (2013 03 28 参照).

[2] 政府の節電ポータルサイト (http://setsuden.go.jp/) (2013 03 28 参照)

[3] ELP Lite (http://www.sassor.com/) (2013 03 28 参照)

[4] Nest Learning Thermostat (http://greenz.jp/2011/11/24/nest-learning-thermostat/) (2013 03 28 参照)

[5] 三菱電機　エレベーター・エスカレーター　AXIEZ : 乗用 [P] 機種一覧・据付 図（ 寸 法 ）(http://www.mitsubishielectric.co.jp/elevator/product/axiez/axiez_p/model.html) (2013 03 28 参照)

[6] 楊　天輝，梶　克彦，河口　信夫，“加速度センサを用いたエレベータ区間検出と移動距離推定”，情報処理学会第 74 回全国大会, pp. 441-442, 2012.

[7] Walkroid - simple pedometer (http://walkroid.kino2718.net/) (2013 03 28 参照)

[8] Ga?tanne Hach?, , Edward D. Lemaire, Natalie Baddour, ‘‘ Wearable Mobility Monitoring Using a Multimedia Smartphone Platform”, IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT, VOL. 60, NO. 9, pp.3153-3164, 2011.

[9] 大久保成晃，菊池浩明, 各種センサを統合した行動状態推定機能を持つライフログシステムの Android への実装と評価, 情報処理学会第 74 回全国大会，6X-2, Vol.3, pp. 35-36, 2012.

[10] 大久保成晃，菊池浩明, DICOMO

謝辞

しゃじしゃじ

付録

状態推定の Java コード

```
1  package net.kiknlab.nncloud.sensor;
2
3  import java.io.File;
4  import java.io.FileOutputStream;
5  import java.io.IOException;
6  import java.text.SimpleDateFormat;
7  import java.util.ArrayList;
8  import java.util.Calendar;
9  import java.util.Date;
10 import java.util.List;
11 import java.util.Locale;
12
13 import net.kiknlab.nncloud.cloud.CloudManager;
14 import net.kiknlab.nncloud.db.DayLogDBManager;
15 import net.kiknlab.nncloud.util.SensorData;
16 import net.kiknlab.nncloud.util.StateLog;
17
18 import android.content.Context;
19 import android.content.SharedPreferences;
20 import android.hardware.SensorManager;
21 import android.os.Environment;
22 import android.preference.PreferenceManager;
23 import android.util.Log;
24 import android.widget.Toast;
25
26 public class StateInference {//状態推定
27   SharedPreferences sp;
28   Context mContext;
29   // 推定変数
30   public StateLog stateLog;
31   private boolean inElevator;
32   public int numSteps;//とりあえず一日の歩数をほじ
33   public int stackStep;//未挿入の歩数
34   public long dayWalk;
35
```

```
36    public String debugLog;
37    public boolean fileExist;
38    public int debugId;
39    public static final String FILE_NAME= "inference_log.csv";
40    public static final String FILE_DIRECTORY = "/NNCloud/";
41
42    // スレッド三回回ってわんわん
43    //final int INTERVAL_PERIOD = 20000;//
          msecかと思ったか，あたりだよ！もう一本！millisecondsです…はい…

44    //Timer timer = new Timer();
45
46    //後から追加したので、あとから整理する
47    //閾値は時間をもとに増減させたいなら、個々のデフォルトに時間をかければいいかなぁ
48    public static final String  TIME_LENGTH              = "
          TIME_LENGTH";
49    public static final long  TIME_LENGTH_DEFAULT          = 5000;//判
          定する時間、デフォルトは五秒間分の値を使う
50    public static final String  TIME_DIFFERENCE_THRESHOLD      = "
          TIME_DIFFERENCE_THRESHOLD";
51    public static final long  TIME_DIFFERENCE_THRESHOLD_DEFAULT =
          1000;//
          msec、加速度と角度のセンサの時間が離れすぎてないことを祈る、祈ってたらこ

52    public static final String  NUMBER_OF_STEPS            = "
          NUMBER_OF_STEPS";
53    public static final String  WALK_COUNT               = "WALK_COUNT
          ";//万歩計っ
          てPedometerだったり、歩数だとNumber of stepsだったり。うーむ
          walk countってだめ？
54    public static final String  WALK_COUNT_DAY            = "
          WALK_COUNT_DAY";
55    public static final String  WALK_COUNT_THRESHOLD        = "
          WALK_COUNT_THRESHOLD";
56    public static final float WALK_COUNT_THRESHOLD_DEFAULT    = 10.4F
          ;//加速
          度, 5.4F
57    public static final String  WALK_THRESHOLD            = "
          WALK_THRESHOLD";
58    public static final int   WALK_THRESHOLD_DEFAULT        = 4;//歩数
59    public static final String  STAIR_THRESHOLD            = "
          STAIR_THRESHOLD";
60    public static final int   STAIR_THRESHOLD_DEFAULT        = 800;//分
          散、あとで閾値調整なぁー
```

```
61    public static final String  ELEVATOR_THRESHOLD           = "
          ELEVATOR_THRESHOLD";
62    public static final long  ELEVATOR_THRESHOLD_DEFAULT       = 1900;
          //msec、
63    public static final String  ELEVATOR_DIFFERENT_INTERVAL     = "
          ELEVATOR_DIFFERENT_INTERVAL_THRESHOLD";
64    public static final int   ELEVATOR_DIFFERENT_INTERVAL_DEFAULT =
          10;//微分する際の
          間隔
65    public static final String  ELEVATOR_DIFFERENT_RANGE       = "
          ELEVATOR_DIFFERENT_RANGE";
66    public static final float ELEVATOR_DIFFERENT_RANGE_DEFAULT  = 0.8
          F;//どこまでの誤差を許容す
          るか
67    public static final String  ELEVATOR_USE_IN            = "
          ELEVATOR_USE_IN";
68    public static final boolean ELEVATOR_USE_IN_DEFAULT        = true;
          //判定時間が短い場合エレベータの開始と終了を同時に取得できないため、現在エレベータ
          の中にいるかどうかを判定する
69    public static final String  ELEVATOR_FEATURE_TIMES         = "
          ELEVATOR_FEATURE_TIMES";
70    public static final int   ELEVATOR_FEATURE_TIMES_DEFAULT     = 1;
          //何回エレベータの特徴を抽出でき
          たか
71
72    public StateInference(Context context) {
73      // ⊂(^ω^ )二二コこいつぁ間に合いそうにないお 2013/03/10
74      // ⊂(^ω^ )二二コこんかいも間に合いそうにないお 2013/03/18
75      // ⊂(^ω^ )二二コやっぱむりだお 2013/03/21
76      // ⊂(;ω; )二二コひぎぃあふぅぅぅぅぅぅ 2013/03/24
77      // ⊂(｀・ω・´)二二コむりだったお 2013/03/25
78      mContext = context;
79      sp = PreferenceManager.getDefaultSharedPreferences(mContext);
80      numSteps = sp.getInt(NUMBER_OF_STEPS, 0);
81      dayWalk = sp.getLong(WALK_COUNT_DAY, getDayTime());
82      stackStep = 0;
83
84      // センサー設定
85      stateLog = new StateLog(StateLog.STATE_STOP);
86      inElevator = false;
87
88      //デバッグログ
89      debugId = 0;
90      debugLog = "";
91      fileExist = true;
```

```
92      File file = new File(Environment.getExternalStorageDirectory()
            + FILE_DIRECTORY);
93      if(!file.exists()){
94        if(!file.mkdir()) fileExist = false;
95      }
96      if(fileExist){
97        file = new File(Environment.getExternalStorageDirectory() +
              FILE_DIRECTORY + FILE_NAME);
98        if(!file.exists()){
99          try{if(!file.createNewFile()) fileExist = false;}
100         catch(Exception e){fileExist = false;}
101       }
102     }
103   }
104
105   public void stop() {
106
107     sp.edit().putInt(NUMBER_OF_STEPS, numSteps).commit();
108   }
109
110   public void inference(List<SensorData> acceles, List<SensorData>
        orientations) {//インファレンスで
        すの
111     Log.e("データ数","[Accele:"+acceles.size()+"]");
112     Log.e("データ数","[Orient:"+orientations.size()+"]");
113     if(acceles.size() <= sp.getInt(ELEVATOR_DIFFERENT_INTERVAL,
          ELEVATOR_DIFFERENT_INTERVAL_DEFAULT) || orientations.size()
          <= 0) return;//値がなければ計算できませ
          んよ
114     //平均とか分散とか垂直加速度とか歩数とか
115     ArrayList<Float> verticalAcceles = calcVerticalAcceleration(
          acceles, orientations);
116     int walkCount = newCountWalk(verticalAcceles);
117     numSteps += walkCount;
118     stackStep += walkCount;
119     overTheDay();
120     sp.edit().putInt(NUMBER_OF_STEPS, numSteps).commit();
121
122     judge(walkCount, acceles, orientations, verticalAcceles);
123   }
124
125   public void judge(int walkCount, List<SensorData> acceles, List<
        SensorData> orientations, ArrayList<Float> verticalAcceles){//
        ジャッジメントで
        すの
```

```
126     //判定するよー！
127     //歩行かそうでないかを判定した後、歩行の場合階段かどうか、歩行でないばあいエレベータの判定を行う
128     debugLog += walkCount + ",";
129     if(judgeWalk(walkCount)){
130       inElevator = false;// エレベータ内では歩かないようお願い申し上げます
131       float[] orientationXZVariance = calcOrientationXZVariance(orientations);
132       debugLog +=  orientationXZVariance[0]+ "," + orientationXZVariance[1] + ",";
133       if(judgeStair(orientationXZVariance[0], orientationXZVariance[1])){
134         stateLog.setStair(acceles.get(acceles.size() - 1).timestamp);
135         debugLog += ",,,," + StateLog.STATE_STAIR;
136       }
137       else{
138         stateLog.setWalk(acceles.get(acceles.size() - 1).timestamp);
139         debugLog += ",,,," + StateLog.STATE_WALK;
140       }
141
142     }
143     else{
144       /*
145       debugLog +=  ",," + ((inElevator)?1:0);
146       if(judgeElevator(verticalAcceles, acceles, inElevator)){
147         inElevator = true;// エレベータ内である
148         stateLog.setElevator(acceles.get(acceles.size() - 1).timestamp);
149         debugLog += "," + StateLog.STATE_ELEVATOR;
150       } else {
151         stateLog.setStop(acceles.get(acceles.size() - 1).timestamp);
152         debugLog += "," + StateLog.STATE_STOP;
153       }
154       */
155       stateLog.setStop(acceles.get(acceles.size() - 1).timestamp);
156       debugLog += ",,,,,," + StateLog.STATE_STOP;
157     }
158
159     SimpleDateFormat sdf = new SimpleDateFormat(CloudManager.DATETIME_PATTERN, Locale.JAPAN);
```

```
160       saveDebugLog(FILE_NAME, debugLog + "," + sdf.format(new Date
              ()), debugId);
161       debugLog = "";
162       debugId++;
163     }
164
165     public boolean judgeWalk(int walkCount) {
166       if (walkCount >= sp.getInt(WALK_THRESHOLD,
              WALK_THRESHOLD_DEFAULT)) {
167         return true;
168       }
169       else  return false;
170     }
171     public boolean judgeStair(float orientationXVariance, float
          orientationZVariance){
172       //あとでSkewness使った判定方式に変更する、きっとする
173       if ((orientationXVariance + orientationZVariance) < sp.getInt(
              STAIR_THRESHOLD, STAIR_THRESHOLD_DEFAULT)) {
174         return true;
175       }
176       else  return false;
177     }
178     public boolean judgeElevator(ArrayList<Float> verticalAcceles,
          List<SensorData> acceles, boolean inElevator){
179       //
              timestampを持ってるのは元データだから引数にとってる。verticalAcceles

180       ArrayList<Float> differentVerticalAcceles = new ArrayList<Float
              >();
181       int differentInterval = sp.getInt(ELEVATOR_DIFFERENT_INTERVAL,
              ELEVATOR_DIFFERENT_INTERVAL_DEFAULT);
182       float differentRange = sp.getFloat(ELEVATOR_DIFFERENT_RANGE,
              ELEVATOR_DIFFERENT_RANGE_DEFAULT);
183       long indexDistinctThreshold = sp.getLong(ELEVATOR_THRESHOLD,
              ELEVATOR_THRESHOLD_DEFAULT);
184       int judgeElevator = 0;
185
186       //微分を行いつつ、最大値と最小値も求める
187       differentVerticalAcceles.add(verticalAcceles.get(
              differentInterval) - verticalAcceles.get(0));
188       float differentVerticalAcceleMax = differentVerticalAcceles.get
              (0);
189       float differentVerticalAcceleMin = differentVerticalAcceles.get
              (0);
```

```
190     for (int i = differentInterval + 1; i < verticalAcceles.size();
            i++) {
191       differentVerticalAcceles.add(verticalAcceles.get(i) -
              verticalAcceles.get(i - differentInterval));
192       if(differentVerticalAcceleMax < differentVerticalAcceles.get(
              i - differentInterval))
193         differentVerticalAcceleMax = differentVerticalAcceles.get(i
                - differentInterval);
194       else if(differentVerticalAcceleMin > differentVerticalAcceles
              .get(i - differentInterval))
195         differentVerticalAcceleMin = differentVerticalAcceles.get(i
                - differentInterval);
196     }
197
198     debugLog += "," + differentVerticalAcceleMax + "," +
            differentVerticalAcceleMin;
199
200     // エレベータ判定
201     //まずエレベータが上昇しているのか下降しているのか考える、この判定方式の残念さは、
            きっと未来の誰かが何とかするさ
202     int featureIndex = differentVerticalAcceles.size();//特徴のインデッ
            クス
203     boolean upElevator = true;//エレベータが下降するとき垂直加速度は増加する、
            逆もまたしかりなかんじで
204     for(int i = 0;i < differentVerticalAcceles.size();i++){
205       if (differentVerticalAcceles.get(i) >
              differentVerticalAcceleMax * differentRange) {
206         featureIndex = i;
207         upElevator = false;
208         break;
209       } else if (differentVerticalAcceles.get(i) <
              differentVerticalAcceleMin * differentRange) {
210         featureIndex = i;
211         upElevator = true;
212         break;
213       }
214     }
215     //debugLog += "," + featureIndex + "," + upElevator;
216     debugLog += "," + ((upElevator)?1:0);
217     for(int i = featureIndex;i < differentVerticalAcceles.size();i
            ++){
218       if(upElevator){
219         if (differentVerticalAcceles.get(i) >
                differentVerticalAcceleMax * differentRange) {
220           if(acceles.get(i + differentInterval).timestamp -
```

```
221                 acceles.get(featureIndex + differentInterval).
                        timestamp >=
222                 indexDistinctThreshold)
223               judgeElevator++;//これバグかな、たぶん
224             featureIndex = i;
225             upElevator = false;
226           }
227         }
228         else{
229           if (differentVerticalAcceles.get(i) <
                differentVerticalAcceleMin * differentRange) {
230             if(acceles.get(i + differentInterval).timestamp -
231                 acceles.get(featureIndex + differentInterval).
                        timestamp >=
232                 indexDistinctThreshold)
233               judgeElevator++;
234             featureIndex = i;
235             upElevator = true;
236           }
237         }
238       }
239       //debugLog += "," + judgeElevator;
240       if (inElevator && sp.getBoolean(ELEVATOR_USE_IN,
              ELEVATOR_USE_IN_DEFAULT) ||
241           judgeElevator >= sp.getInt(ELEVATOR_FEATURE_TIMES,
                  ELEVATOR_FEATURE_TIMES_DEFAULT)) {
242           if(inElevator) inElevator = false;
243         return true;
244       }
245       else return false;
246     }
247
248     //基準から、超えている間と未満の間、それぞれの最大最小値を比較
249     //基準から、超えている間と未満の間、それぞれの平均値を比較m
250     //重力と、平均値それぞれを基準に使った方法
251     //合成加速度、垂直加速度それぞれを使った方法
252     private float NEW_WALK_COUNT_FIRST_THRESHOLD = 0;
253     private float NEW_WALK_COUNT_SECOND_ACCELE_THRESHOLD = 0.95f;
254     public int newCountWalk(ArrayList<Float> verticalAcceles) {
255       //9.8を前後するかどうか
256       int count = 0;
257       float acceleHigh = verticalAcceles.get(0);
258       float acceleLow = verticalAcceles.get(0);
259       boolean upStep = true;
260       float avg = 0;
```

```
261
262     for(float accele : verticalAcceles){
263       if(upStep){
264         if(accele > acceleHigh) acceleHigh = accele;
265         if(accele >= SensorManager.GRAVITY_EARTH + this.
                NEW_WALK_COUNT_FIRST_THRESHOLD){
266           upStep = false;
267           acceleLow = accele;
268         }
269       }else{
270         if(accele < acceleLow)  acceleLow = accele;
271         if(accele <= SensorManager.GRAVITY_EARTH + this.
                NEW_WALK_COUNT_FIRST_THRESHOLD){
272           upStep = true;
273           //Log.e("Walk",acceleHigh - acceleLow + "");
274           avg += acceleHigh - acceleLow;
275           if(acceleHigh - acceleLow >
                NEW_WALK_COUNT_SECOND_ACCELE_THRESHOLD){
276             count++;
277           }
278           acceleHigh = accele;
279         }
280       }
281     }
282     //Toast.makeText(mContext.getApplicationContext(), avg + "",
           Toast.LENGTH_SHORT).show();
283     return count;
284   }
285   public int countWalk(ArrayList<Float> verticalAcceles) {
286     int walkCount = 0;
287     boolean upWalk = true;//加速度が上昇しているか
288     float acceleHigh = verticalAcceles.get(0);
289     float acceleLow = verticalAcceles.get(0);
290     int i = 0;
291
292     //加速度の増加減少の勢いを見て、閾値を超えた場合一歩とカウントする
293     for(float accele : verticalAcceles){
294       if (upWalk) {//加速度が上昇時
              にHighを更新、下降時にLowを更新して下降判定に移る
295         if (acceleHigh <= accele) acceleHigh = accele;
296         else {
297           //debugLog += "up," + i + "," + acceleHigh + "," +
                  acceleLow + "," + (acceleHigh - acceleLow) + "\n";
298           //if (acceleHigh - acceleLow > sp.getFloat(
                  WALK_COUNT_THRESHOLD, WALK_COUNT_THRESHOLD_DEFAULT))
```

```
                walkCount++;
299             acceleLow = accele;
300             upWalk = false;
301           }
302         } else {// 下降時も似た感じで
303           //debugLog += "down," + i + "," + acceleHigh + "," +
                acceleLow + "," + (acceleHigh - acceleLow) + "\n";
304           if (acceleLow >= accele) acceleLow = accele;
305           else {
306             if (acceleHigh - acceleLow > sp.getFloat(
                  WALK_COUNT_THRESHOLD, WALK_COUNT_THRESHOLD_DEFAULT))
                  walkCount++;
307             acceleHigh = accele;
308             upWalk = true;
309           }
310         }
311         i++;
312       }
313       return walkCount;
314     }
315
316     public float[] calcOrientationXZVariance(List<SensorData> datas){
317       //計算量減らすために汎用性を犠牲にした。もっといい方法があるはずだし、使いまわせな
            い関数って嫌いなので、任せたぜ未来のオレ
318       float avg1 = 0, avg2 = 0;
319       float variance1 = 0, variance2 = 0;
320       for(SensorData data : datas){
321         //Log.e("inference","[X:" + Math.toDegrees(data.values[1]) +
              "]");
322         avg1 += Math.toDegrees(data.values[1]);
323         avg2 += Math.toDegrees(data.values[2]);
324         variance1 += Math.toDegrees(data.values[1]) * Math.toDegrees(
              data.values[1]);
325         variance2 += Math.toDegrees(data.values[2]) * Math.toDegrees(
              data.values[2]);
326       }
327       avg1 = avg1 / datas.size();
328       avg2 = avg2 / datas.size();
329       variance1 = (variance1 / datas.size()) - (avg1 * avg1);
330       variance2 = (variance2 / datas.size()) - (avg2 * avg2);
331       return new float[]{variance1, variance2};
332     }
333     public float[] calcAverageAndVariance(float[] datas){
334       float avg = 0;
335       float variance = 0;
```

```
336     for(float data : datas){
337       avg += data;
338       variance += data * data;
339     }
340     avg = avg / datas.length;
341     variance = (variance / datas.length) - (avg * avg);
342     return new float[]{avg, variance};
343   }
344
345   public ArrayList<Float> calcVerticalAcceleration(List<SensorData>
        accele,List<SensorData> orientation){// 垂直加速度の配列の計
        算
346     ArrayList<Float> verticalAcceles = new ArrayList<Float>();
347     int orientationIndex = 0;
348     boolean checkIndex = true;
349     try {
350     for(int i = 0;i < accele.size();i++){
351       while(checkIndex){//加速度の取得時と回転度の取得時がずれすぎていないかを
            チェック、もっといい方法あるよなぁ、そのうち改善する…
352         if(Math.abs(orientation.get(orientationIndex).timestamp -
              accele.get(i).timestamp) <
353               Math.abs(orientation.get(orientationIndex+1).timestamp
                  - accele.get(i).timestamp)){
354           break;
355         }
356         orientationIndex++;
357         if(orientationIndex >= (orientation.size() - 1)){
358           if(Math.abs(orientation.get(orientationIndex).timestamp -
                accele.get(i).timestamp) >
359           Math.abs(orientation.get(orientationIndex-1).timestamp -
                accele.get(i).timestamp)){
360             orientationIndex--;
361           }
362           checkIndex = false;
363         }
364       }
365       verticalAcceles.add(calcVerticalAcceleration(accele.get(i),
            orientation.get(orientationIndex)));
366     }
367     } catch(Exception e){}
368     return verticalAcceles;
369   }
370   public float calcVerticalAcceleration(SensorData accele,
        SensorData orientation){// 垂直加速度の
        SensorData型からfloatにして計算
```

```
371     return calcVerticalAcceleration(accele.values, orientation.
            values);
372   }
373   public float calcVerticalAcceleration(float[] accele,float[]
          orientation){// 垂直加速度の計算
374     //加速度三軸をXYZの順で、角度はYXZの順かつラジアンでおなしゃーす
375     float verticalAccele = (float)(
376         accele[0] * Math.cos(orientation[1]) * Math.sin(orientation
                [2])
377         - accele[1] * Math.sin(orientation[1]) * Math.cos(
                orientation[2])
378         + accele[2] * Math.cos(orientation[1]) * Math.cos(
                orientation[2]));
379     return verticalAccele;
380   }
381
382   public static void saveDebugLog(String fileName, String logs, int
          id){
383     logs = logs + "\n";
384     FileOutputStream fos = null;
385
386     try {
387       fos = new FileOutputStream(Environment.
              getExternalStorageDirectory() +
388           FILE_DIRECTORY +
389           fileName,true);
390       fos.write(logs.toString().getBytes(), 0, logs.length());
391       fos.flush();
392     }
393     catch (Exception e) {e.printStackTrace();Log.e("inferenceFile",
            "exception1");}
394     finally {
395       try {
396         if( fos != null ){
397           fos.close();
398         }
399       }
400       catch( IOException e ){e.printStackTrace();Log.e("
              inferenceFile","exception1");}
401     }
402   }
403
404   public void overTheDay(){//今日を越えて
405     if(java.lang.System.currentTimeMillis() >= dayWalk + (24 * 60 *
            60 * 1000)){
```

```
406         //DayLogDBManager.insertStepLog(mContext, numSteps, dayWalk);
407         numSteps = 0;
408         dayWalk = getDayTime();
409         sp.edit().putLong(WALK_COUNT_DAY, dayWalk).commit();
410         }
411     }
412     public long getDayTime(){
413       Calendar calendar = Calendar.getInstance();
414       calendar.set(calendar.get(Calendar.YEAR), calendar.get(Calendar
              .MONTH), calendar.get(Calendar.DATE), 0, 0, 0);
415       calendar.add(Calendar.MILLISECOND, -calendar.get(Calendar.
              MILLISECOND));
416       return calendar.getTimeInMillis();
417     }
418 }
```