平成27年度　卒業論文

和文題目

スマートフォンアプリケーションの消費電力低減の提案と評価

英文題目

Proposal of the Reduction of Power Consumption in Smartphone Application and its Evaluation

情報工学科　渡邊研究室

(学籍番号: 120430066)

坪井　俊也

提出日: 平成 28 年 2 月 10 日

名城大学理工学部

概要

少子高齢化と核家族化の進行により高齢者の徘徊行動や孤独死などが問題視されている．我々は，スマートフォンの通信機能とセンサ機能を活用し，見守る側（家族や地域の人など）と見守られる側（高齢者や子どもなど）で，ユーザの位置情報や行動状態などの情報を共有することにより，住民が安心して生活できるシステムとして統合生活支援システム TLIFES（Total LIFE Support system）[1,2] を提案している．TLIFES で使用するスマートフォン側アプリケーションは，その消費電力が稼働時間に大きな影響を与える．これまでの省電力化の取り組みにより消費電力は小さくなりつつあるが，加速度センサを用いた歩数計，行動状態の判定は，依然として消費電力が大きく，見守りシステムとして大きな課題となっている．そこで本稿では，消費電力の大きい加速度センサ処理に着目し，消費電力低減方法の提案を行った．提案方式を Android 上に実装して評価を行い，CPU による消費電力をこれまでの約 40%に低減できることを確認した．

目 次

第1章　序論

超高齢化社会の到来，核家族化の進行により，1 人暮らしの高齢者の増加や高齢者の徘徊行動が社会問題となっている．警察庁によれば 70 歳以上の行方不明者は年間 1 万 5 千人を超えており，認知症の行方不明者は年間 1 万人を超えている [3]．また，10 歳代の行方不明者は年間 1 万 7 千人以上報告されており [3]，子供の行方不明，誘拐事件も社会問題のひとつとなっている．そのため，高齢者や子供がどこに居ても見守ることができる仕組みが求められている．

一方で，Android などといった GPS や無線通信，加速度センサ，地磁気センサを搭載したスマートフォンが急速に普及してきており，誰もが手軽に手に取り利用できるようになっている．

そこで我々は，スマートフォンとモバイルネットワークを利用した生活支援システム TLIFES（Total LIFE Support system）[1, 2] を提案している．TLIFES は，ユーザがスマートフォンを所持することを前提としており，見守る側（家族や地域の人など）と見守られる側（高齢者や子どもなど）で情報を共有し，異常を警報することにより見守りを実現している．見守りは，スマートフォンに搭載されている GPS，加速度センサなどを利用して得られたユーザの位置や行動状態などの情報を定期的にサーバへ送信し，過去のデータとともに蓄積・解析することにより異常検出を行っている．

TLIFES の課題のひとつにスマートフォンの消費電力量がある．特に GPS はセンサの中でも消費電力は非常に大きいが，TLIFES では見守りのため定期的に GPS を起動させる必要がある．そのためスマートフォンの稼働時間が短いという，見守りシステムとして重要な課題となっている．これまでの TLIFES [1,4] では，屋内判定により GPS 起動時間を抑える，加速度値を用いたユーザの行動判定により,「放置中」「静止中」「乗車中」「歩行中」の 4 種類を判定し，移動中と判定したときにのみ GPS を起動させるといった工夫により電力を低減させた．さらに,「放置中」には端末をスリープさせ最小限の処理のみを行うようにした．これらの取り組みにより，電力消費は大きく削減されてきたものの，行動状態の判定をするために，常に加速度センサを利用する必要があり，CPU が動作することによる消費電力の割合が依然として大きい点が課題となっていた．

そこで本稿では，消費電力の大きい加速度センサの CPU 処理に着目した消費電力低減方法として，以下の 2 つを提案する．第一に加速度センサ処理部を Java 言語から C 言語へ書き換えることにより，CPU の無駄な処理を除去し，処理時間を短縮する．第二に加速度センサ取得間隔を拡大し，必要最小限のデータにより処理を実行する．提案を Android 上に実装した結果，CPU による消費電力をこれまでの約 40%に低減できることを確認した．

以下，2 章において TLIFES の概要，3 章において従来の TLIFES の消費電力低減方法と課題について述べる．4 章において提案方式について，5 章において提案方式を実装した部分について述べる．6 章において提案方式の評価方法・結果について述べ，7 章において結論を述べる．

第2章　TLIFES

図 1 に TLIFES の概要を示す．TLIFES では，すべてのユーザがスマートフォンを所持していることを前提とする．スマートフォンの通信機能とセンサ機能を活用し，ユーザ同士が情報を共有することができる．センサ情報の取得には，GPS や加速度センサを用いる．スマートフォンは，取得したユーザの位置情報や行動情報をインターネット上の TLIFES サーバに定期的に送信し，データベースに蓄積する．蓄積された情報は，許可されたユーザであればいつでも閲覧することができる．TLIFES サーバでは，スマートフォンから随時送信されてくる情報と，既に蓄積されている情報を比較することにより，徘徊行動や誘拐などユーザに異常がないかどうかを判断する．異常が検出された場合には，予め登録されたメールアドレスに対し，アラームメールを配信する．これにより，緊急時においても早期に対応が可能となる．また，ユーザも自身のセンサ情報を閲覧し，私生活や健康管理について振り返ることにより，ライフログとしても活用できる．

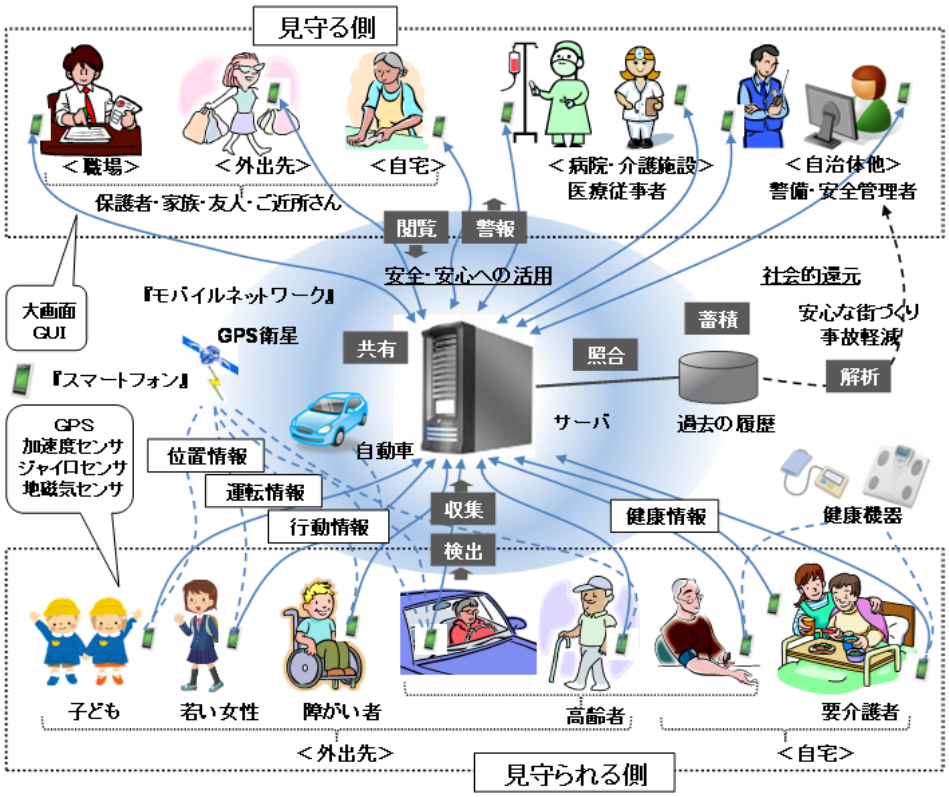


図 1　TLIFES の概要

第3章　これまでの消費電力低減対策

TLIFES では，スマートフォンアプリケーションの消費電力を減らすことが重要である．特に GPS はセンサの中でも消費電力は非常に大きく，GPS の起動をできるだけ抑える必要がある．本章では，スマートフォンアプリケーションの省電力化のために，TLIFES でこれまでどのような対策がなされてきたかを述べる [1,4].

3.1　GPS による屋外・屋内判定

GPS は，屋内のような GPS 衛星の電波が届かない場所においても，必要以上に位置情報を取得しようとして電力を消費する場合がある．このとき，大きな誤差を含んだ位置情報を取得する場合がある．このような状況に対応するため，捕捉している GPS 衛星数から屋内・屋外の判定を行う．GPS は衛星を 4 機以上捕捉しないと正確な位置情報を取得できない．捕捉衛星数は比較的早く検出できるため，測位開始 10 秒後に GPS 衛星数を 4 機以上補足できなかった場合，屋内にいて位置情報が取得できないと判定し，GPS 測位を中断するようにした．GPS 捕捉衛星数が 4 機以上の場合を屋外と判定し位置測位を続ける．この判定により，室内では即座に測位を終了させることにより，GPS 起動時間を少なくさせ，消費電力は大きく低減された．

3.2　加速度値によるユーザの行動状態判定

自宅や職場などで長時間移動しない場合は，GPS を起動する必要がない．そのため，加速度センサの 3 軸合成値を用いた保持判定，歩数計，乗車判定により,「放置中」「静止中」「乗車中」「歩行中」の 4 種類のユーザの行動状態を判定し，移動中と判定したときにのみ GPS を起動させることとした．また,「放置中」には端末をスリープさせ最小限の処理のみを行うようにした．加速度センサはセンサ類の中で消費電力が小さく，情報を取得するときの場所に依存することがないことが利点である．

図 2 に加速度センサを用いた行動判定のフローチャートを示す.

- 保持判定
  サーバへの定期送信間である 2 分間，加速度センサ値に変化がない場合，ユーザがスマートフォンを保持せず，放置されていると判断し,「放置中」と判定する．加速度センサ値に変化がある場合は，ユーザがスマートフォンを保持していると判断し，歩数を用いた歩行判定を行う．

- 歩行判定

  2 分間の歩数を計測する．その歩数が 60 歩/分 以上である場合，ユーザが歩行して移動していると判断し,「歩行中」と判定する．60 歩/分 未満の場合，乗車判定を行う.「歩行中」と判定された場合，ユーザが移動しているため GPS を起動し，位置情報の取得・更新を行う．

- 乗車判定

  車や電車などの乗り物に乗車している時に，加速度センサで検出することのできる高周波の振動を利用して判定を行う．加速度値の 2 乗平均値が一定値以上の場合，ユーザが何らかの乗り物に乗車していると判断し,「乗車中」と判定する．加速度値の 2 乗平均値が一定値未満の場合，ユーザがスマートフォンを所持しているが静止していると判断し,「静止中」と判定する.「乗車中」と判定された場合，ユーザが移動しているため GPS を起動し，位置情報の取得・更新を行う．

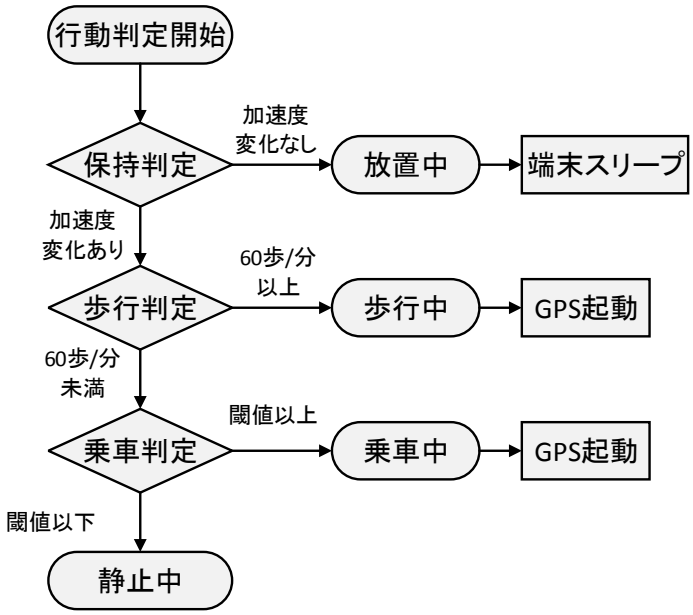


図 2　加速度センサを用いた行動判定のフローチャート

図 3 に 20ms ごとの処理と 2 分ごとの処理の関係を示す．現在の TLIFES アプリケーションの加速度センサ取得間隔は 20ms であるため，加速度センサイベントや加速度変化判定，歩数計は 20ms ごとに行われる．また，サーバへの定期送信間隔は 2 分間のため，図 2 にも示されている行動判定処理，保持判定，歩行判定，乗車判定は 2 分ごとに行われる．20ms ごとの処理によって求めた加速度値の変化や歩数を，2 分ごとに受け取り行動判定に用いて各判定を行う．

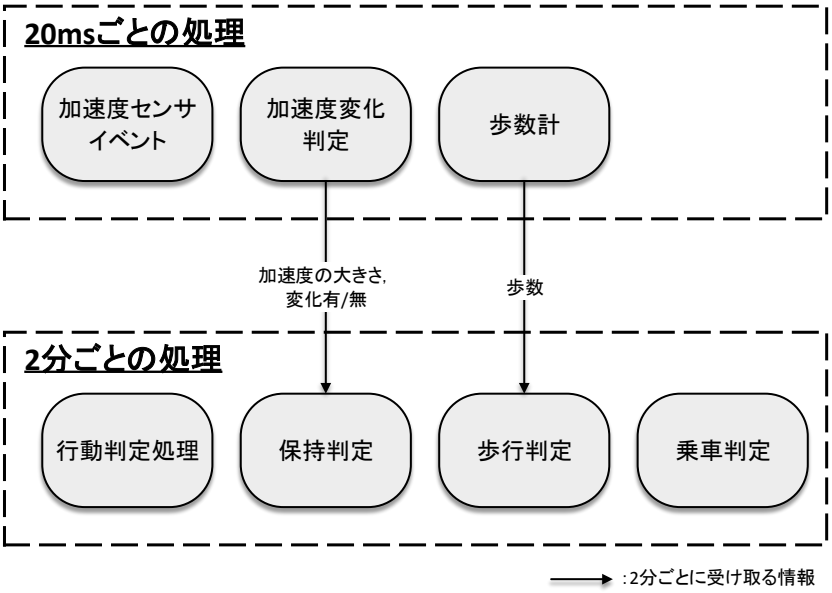


図 3　20ms ごとの処理と 2 分ごとの処理の関係

3.3　課題

以上の取り組みにより，GPS にかかる電力消費は大きく削減された．しかし，ユーザの行動状態の判定をするために，CPU が常に加速度センサ値を利用した処理を行う．そのため，CPU の動作にかかる消費電力の割合が依然として大きいという課題がある．そこで，CPU 消費電力を重点的に低減する対策が必要である．

第4章　提案方式

常に動作している加速度センサ値の CPU 処理を重点的に軽減することにより，アプリケーションの消費電力を大きく低減できる．そこで，以下の 2 通りの方法の提案する．

4.1　提案 A：加速度センサ処理部の C 言語書き換え

現在の TLIFES アプリケーションは Android 上に実装されているため，すべて Java 言語により記述されている．文献 [5] によれば，Java 言語実装と比較して，C 言語実装とした場合，実行時間が短く，消費電力が小さいとされている．そのため，一部機能を C 言語に書き換えることにより消費電力低減が図れる．そこで Java 言語空間と C 言語空間とのコード記述の線引き，書き換えを行う部分を検討した．

まず，消費電力の大きな要因となっている，図 3 に示す 20ms ごとに動作している加速度変化判定と歩数計を，C 言語に書き換えることとした．次に，Java 言語空間と C 言語空間の遷移をできるだけ無くす必要がある [6]．そこで，加速度センサイベントからユーザの行動状態判定までの一連の処理も含めて，すべて C 言語により記述することとした．これにより，加速度センサ値の取得から一連の処理までを C 言語空間によって閉じることができる．Java 言語は 2 分ごとの処理として，保持判定結果，歩数，乗車判定結果を取得する．これにより Java 言語と C 言語間の無駄な遷移を無くした．

以上の検討より，加速度センサイベントと，保持判定，歩数計，乗車判定の各処理をすべて C 言語に書き換えることとした．

4.2　提案 B：加速度センサ取得間隔の拡大

現状 20ms の加速度センサ取得間隔を大きくすることにより CPU 計算回数を減らし，消費電力低減が図れる．加速度センサ取得間隔を大きくしすぎるとナイキスト周波数 [*1] が小さくなり，ユーザの行動状態の誤判定の原因になる．そこで行動状態判定に必要となる最大の取得間隔を検討した．

行動判定に必要な加速度値の周波数成分の解析を行った．方法は，TLIFES で実際に使用している 2 分間の加速度値について，端末をズボンの右ポケットに入れて実測し，周波数解析を行った．図 4 に行動ごとの加速度値の周波数解析結果を示す．行動判定と周波数解析の結果を以下に示す．

[*1] アナログ信号をサンプリングするとき，そのサンプリング周波数の 1/2 となる周波数．サンプリングの際，ナイキスト周波数よりも小さい周波数成分は，デジタル化したデータから元のアナログ信号に正確に復元できる．

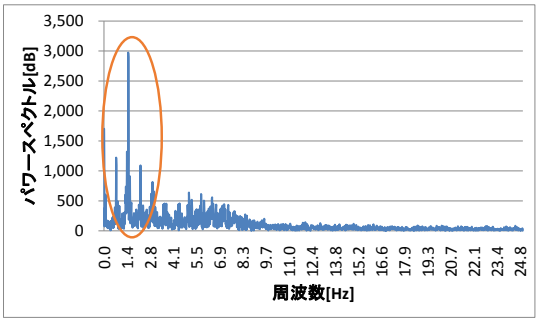
- 保持判定

  加速度値が極めて小さいため，周波数成分を判定に利用する必要がない．
- 歩行判定

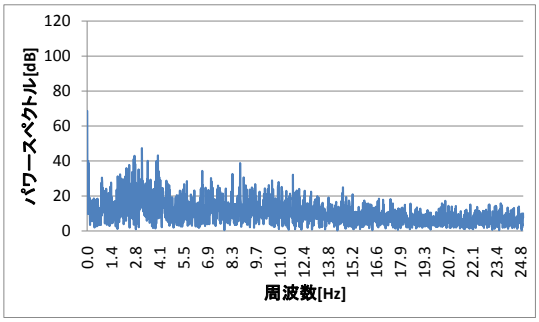
  人間は 1 秒間に 2 歩程度歩くため，必要となる周波数は 2Hz 前後である（図 4a）．
- 静止判定

  静止中は特徴的な周波数はない（図 4b）．
- 乗車判定

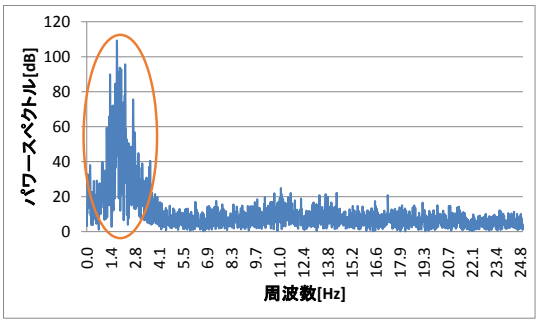
  乗車中における加速度値は，JR は 2Hz 前後（図 4c），車は 1Hz～8Hz 程度（図 4d）の周波数成分が大きい．



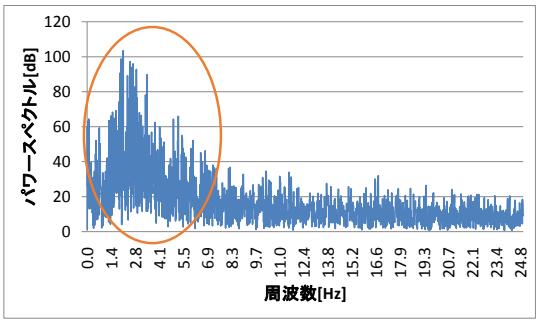
(a) 歩行時



(b) 静止時



(c) JR 乗車時



(d) 車乗車時

図 4　行動ごとの加速度値の周波数解析結果

以上より，ナイキスト周波数は 8Hz 程度でよく，$\frac{1}{8[Hz]\times 2} = 0.0625[s]$ より，サンプリング間隔は 60ms で十分である．したがって，加速度センサ取得間隔を従来の 20ms から 60ms まで大きくしても，行動判定結果の認識率を維持したまま消費電力が低減できる．

第5章　実装

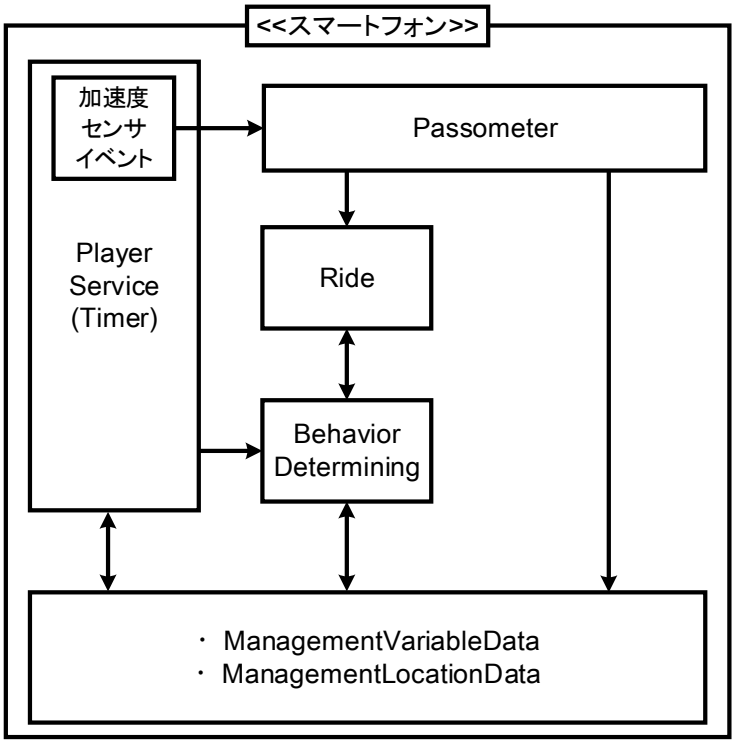
提案方式を Samsung Galaxy Nexus （Android4.2）上に実装した．

5.1　加速度センサ処理部の C 言語書き換え

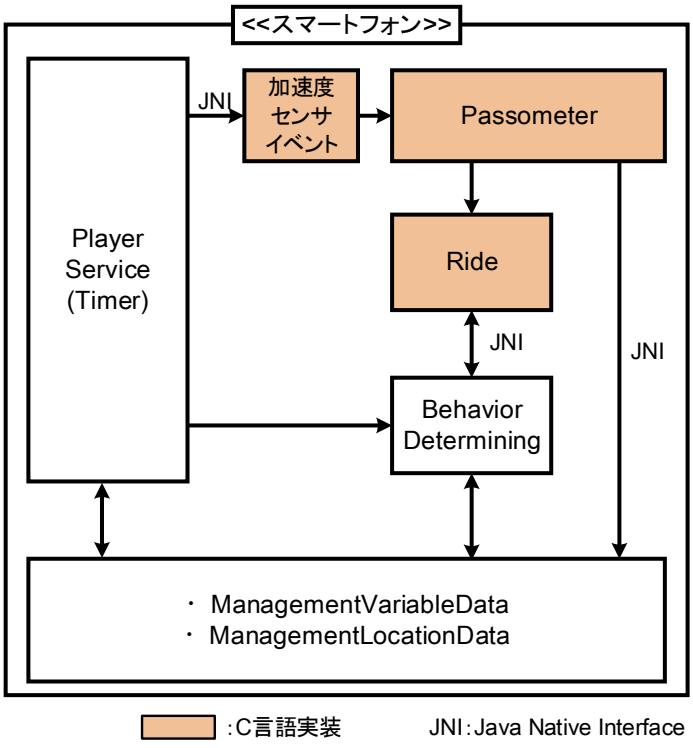
提案 A である，加速度センサイベントと保持判定，歩数計，乗車判定の各処理をすべて C 言語に書き換え，実装した．

Android アプリケーション上での C 言語による実装は，Java から JNI（Java Native Interface）と呼ばれる，Java コードと C/C++言語で書かれたネイティブコードを連携させる仕組みを利用して行った．この仕組みを用いることにより，Java のメソッドを呼び出すときと同じように，C/C++言語によって実装された関数を呼び出すことができる．

図 5 に，従来のモジュール構成と提案方式のモジュール構成の比較を示す．従来（図 5a）は，すべて Java 言語により実装されている．加速度センサイベント，Passometer は 20ms ごと，Ride, Behavior Determining は 2 分ごとの処理となっている．提案方式では，図 5b 中の色塗りされている，加速度センサイベントと Passometer，Ride を C 言語に書き換え実装した．C 言語と Java 言語とのデータの受け渡しは JNI を用いて行う．



(a) 従来のモジュール構成図



(b) 提案方式のモジュール構成図

図 5　従来のモジュール構成と提案方式のモジュール構成の比較

各モジュールの機能は以下のとおりである．

- 加速度センサイベント
  加速度センサから加速度値を指定した取得間隔で取得を行う．Player Service より加速度センサの ON/OFF を受けて動作を開始/終了させる．取得した x,y,z 軸の加速度値とタイムスタンプを Passometer へと送る．図 3 中の ”加速度センサイベント ” にあたる．
- Passometer
  加速度センサイベントより，加速度センサから加速度値を取得し，歩数を計算する．3 軸加速度の合成，バターワースフィルタ処理によるノイズの低減，歩数のカウントや加速度変化判定を行う．また，乗車判定に利用する 2 分間の 3 軸合成後の加速度値を格納して，2 分に 1 回 Ride に送る．さらに，Management Variable Data・Management Location Data へと，定期的に歩数や加速度変化判定結果の情報を送る．図 3 中の ”加速度変化判定 ”，”歩数計 ” にあたる．
- Ride
  Behavior Determining からの乗車判定呼び出しにより，Passometer が格納した 2 分間の 3 軸合成後の加速度値を取得し，乗車判定を行う．ノイズ除去のための軸調節の処理，フィルタ処理，振幅制限の処理を行い，ノイズ除去後の加速度値の 2 乗平均値を算出する．算出した 2 乗平均値により静止中と乗車中の判定を行う．乗車判定結果を Behavior Determining に返す．図 3 中の ”乗車判定 ” にあたる．
- Behavior Determining
  2 分ごとに，加速度値を処理した情報より行動判定を行う．Management Variable Data・Management Location Data から，スマートフォンの加速度変化判定，歩数を取得して判定を行う．「放置中」,「歩行中」でなかった場合に乗車判定を行うため，Ride を呼び出し，乗車判定の結果を取得する．また,「歩行中」,「乗車中」であった場合に GPS による位置情報更新要求を行う．図 3 中の ”保持判定 ”，”歩行判定 ”，”行動判定処理 ” にあたる．
- Management Variable Data・Management Location Data
  歩数や行動判定の結果，位置情報などを格納する．

表 1 に従来と提案方式，提案にかかる変更部分の各ステップ数 [⋆1] を示す．ここで，総ステップ数はプログラムのソースコードの総行数であり，実ステップは総ステップのうち空行やコメント行を除いたソースコードの行数である．提案方式では，アプリケーション全体の約 1 割の書き換えを行った．

[⋆1] プログラムの規模を測るための指標

表 1　従来と提案方式，提案にかかる変更部分の各ステップ数

	総ステップ数	実ステップ数
従来	10,589	6,371
・内書き換え/修正	877	597
提案方式	11,278	6,719
・内修正/追加	1,367	821

5.2　加速度センサ取得間隔の拡大

提案 B である，加速度センサ取得間隔を 20ms から 60ms に変更した．

変更した部分としては，図 5b 中の加速度センサイベントにおけるセンサ取得間隔を 20ms としていたところを 60ms とした．また，加速度センサ取得間隔の変更に影響する歩数計のバンドパスフィルタを 60ms 対応のものとした．その他，20ms に依存して決められていた定数を変更して実装を行った．

第6章　評価

以上の提案を Android 上に実装し，アプリケーションの消費電力量を測定して，従来方式と提案方式との比較評価を行った．

6.1　提案方式の行動判定認識率

消費電力測定の前に，従来方式と提案方式の行動判定の認識率を測定して，提案方式の認識率が従来と変わらないことを確認した．

測定方法・条件を以下に示す．

- 使用端末：Samsung Galaxy Nexus （Android4.2） 2 台
- 一方の端末で従来方式のアプリケーションを動作させて，もう一方の端末で提案方式のアプリケーションを動作させた．
- 2 台の端末の時刻を合わせ，同じタイミングでアプリケーションを起動させた．
- 2 台を重ねてズボンの右ポケットに入れ，できるだけ同じ振動が入力されるようにした．
- 端末の個体差による偏りを少なくするために，従来方式と提案方式のアプリケーションを起動させる端末を，時折，入れ替えた．
- 認識率を求めるため，測定時の実際の行動を記録した．

表 2 に従来方式と提案方式の行動判定の認識率を示す．4 種類の行動状態において，従来方式と提案方式でそれぞれの行動判定の認識率を求めた．

表 2 の結果より，従来方式と提案方式との認識率の差は小さい．これにより，提案方式の行動判定の認識率が従来と変わらないことを確認した．

表 2　従来方式と提案方式の行動判定の認識率

	従来方式の認識率 [%]	提案方式の認識率 [%]
放置中	100.0	100.0
静止中	85.8	86.4
乗車中	73.4	71.1
歩行中	95.8	95.8

6.2 消費電力測定方法

測定に用いた端末と Android アプリケーションを以下に示す.

- 使用端末：Samsung Galaxy Nexus （Android4.2） 1 台
- 測定アプリケーション：CORE Power Profiler [7]

CORE Power Profiler は Android スマートフォンの消費電力を詳細に測定できるアプリケーションである.

6.3 消費電力測定条件

消費電力測定は，従来方式と，提案方式を実装したアプリケーションを個々に作成して，C 言語実装の有無の 2 通り，および加速度センサ取得間隔（20ms と 60ms）の 2 通りを，ユーザの行動判定 4 通り（放置中，静止中，乗車中，歩行中）ごとに計 16 項目ついて行った.

測定は，すべての項目について同一の端末を用いて，それぞれ独立した時間に行った.

CORE Power Profiler は 1 時間単位で消費電力を測定可能なため，4 つの行動判定結果ごとに，3 時間ずつ測定して 1 時間の平均値を測定結果に用いた．しかし，3 時間常に，乗車していることや，歩行していることは難しい．またユーザの行動判定結果には誤判定の可能性もある．測定にはアプリケーションの処理として，図 2 のような，行動判定フローチャートの分岐，処理が正しく行えていればよい．そのため，実際に乗車，歩行する必要はなく，図 4 のような行動ごとの特徴的な周波数を持つ振動を sin 関数により与えることによって，3 時間常に,「乗車中」「歩行中」などの状況を実現した.

アプリケーションの動作としては，測定の 1 時間で 30 回行動判定を行い,「乗車中」「歩行中」の判定につき，GPS が約 10 秒間起動する.

6.4 比較評価結果

図 6 に従来方式と提案方式の消費電力比較を示す．以下の 4 通りの実装状態における放置中，静止中，乗車中，歩行中ごとの，計 16 項目の TLIFES 消費電力（単位：[mAs/h]）となっている.

- 従来方式 ”Java：20ms”
- 提案 B のみを実装 ”Java：60ms”
- 提案 A のみを実装 ”Java+C：20ms”
- 提案 A・B を両方実装 ”Java+C：60ms”

1 項目における消費電力の内訳は，”CPU”，”GPS”，”Accelerometer”からなる.

- ”CPU”：アプリケーションの CPU 処理にかかる消費電力
- ”GPS”：GPS のハードウェアにかかる消費電力
- ”Accelerometer”：加速度センサ自体にかかる消費電力

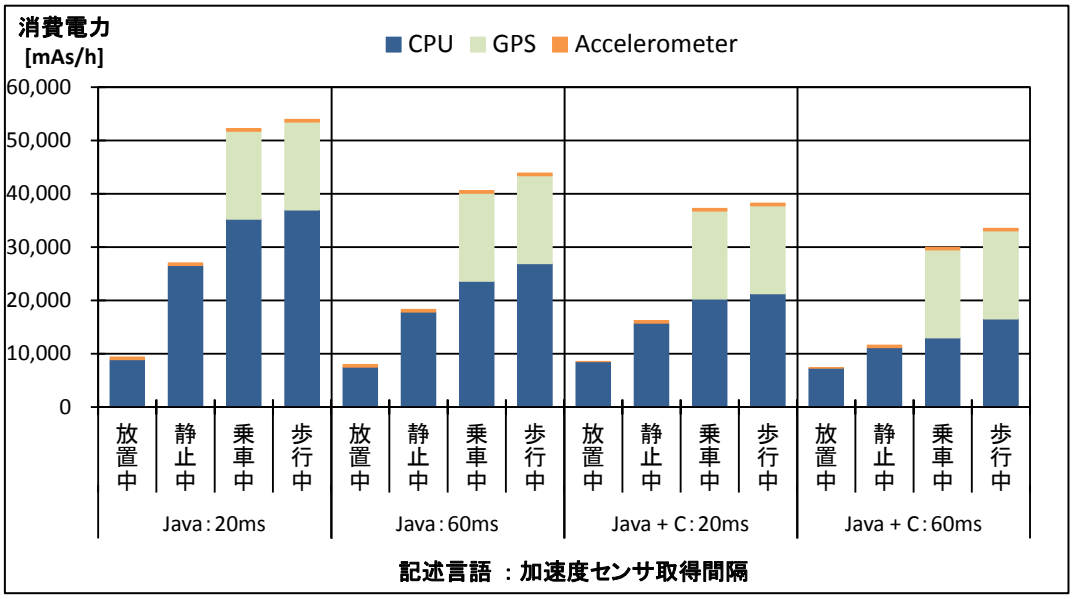


図 6　従来方式と提案方式の消費電力比較

例えば，従来方式での「放置中」における消費電力は，約 10,000[mAs/h] と読む．

「静止中」の消費電力は従来方式（Java : 20ms）と比較して，加速度センサ間隔を 60ms とした場合（Java : 60ms）は 67.7%，センサ処理部を C 言語実装とした場合（Java+C : 20ms）は 60.0%，2 つの提案をともに実装した場合（Java+C : 60ms）は 43.0%となった．

「乗車中」「歩行中」の消費電力は，GPS の電力があるため 2 つの提案をともに実装した場合は，それぞれ従来方式の 57.4%，62.1%となったが，CPU 処理に着目するとそれぞれ従来方式の 36.8%，44.7%となった．

「放置中」の消費電力は，2 つの提案をともに実装した場合は従来方式の 78.5%となった．「放置中」において，他の 3 つの行動状態と比較して，提案方式に対して消費電力低減効果が薄い．これは，「放置中」判定時には CPU をスリープさせ，最低限の処理のみを行うようにしており，提案方式による CPU 処理量の低減の効果があまり得られなかったためである．

本提案において，アプリケーションソースコードの約 1 割に変更を加え，重点的に消費電力低減を図ってきた CPU 処理は，元より電力の小さい「放置中」を除けば，従来方式の約 40%となる結果が得られた．

6.5　今後の課題

提案方式では，CPU 処理の低減に成功したが，CPU は放置中以外，常に起動している．そのため，CPU がスリープせず常に起動していることによる基礎消費電力は依然大きい点が今後の課題と考える．

そこで，Android4.4 以降かつ，端末が対応していることで，利用可能となった Hardware Sensor Batching [8] という機能を利用することにより，CPU をスリープさせたままハードウェアがセンサ

値を取得し，定期的に一度にソフトウェア側にセンサイベントを起こすことが可能となる．これを適用することにより，従来と同じ処理・動作を保ちながら，CPU スリープの時間を確保できるため，更なる消費電力低減が見込めると考える．

第7章　結論

本稿では，スマートフォンアプリケーションの消費電力低減方法を提案した．従来の TLIFES アプリケーションにおいて，ユーザの行動状態の判定をするために，常に加速度センサを利用するため，CPU の動作に対する消費電力の割合が大きい点が課題となっていた．そこで，CPU 処理に着目した消費電力低減方法として，加速度センサ処理部を C 言語に書き換えることと，加速度センサ取得間隔を大きくすることを提案した．提案方式では，加速度センサイベントの取得から行動判定までの一連の処理を，すべて C 言語により記述して，Java 言語と C 言語間の遷移と最小限とした．また，歩行や乗車などの行動ごとの加速度値の周波数解析を行うことにより，適切な加速度センサ取得間隔を求めた．2 つの提案を Android のスマートフォンに実装し，アプリケーション消費電力を測定することにより提案方式の評価も行った．その結果，提案方式は CPU による消費電力を従来方式の約 40%に低減できることが示された．

謝辞

本研究を行うにあたり，研究の方向性など多大なる御指導とご教授を賜りました指導教官の渡邊晃教授に心より感謝いたします.

本研究を進めるにあたり，常日頃からご意見ならびにご助言を受け賜りました，TLIFES 関係者の皆様に深く感謝をしております.

最後に，本研究を行うにあたり，本研究室の皆様にも多くの方々から多大な助言と協力を受け賜り，深く感謝しております.

参考文献

[1] 加藤大智，竹腰昇太，大野雄基，鈴木秀和，旭　健作，渡邊　晃：TLIFES における省電力化を目的とした位置測位手法の提案と実装，研究報告コンシューマ・デバイス&システム (CDS), Vol. 2013-CDS-6, No. 13, pp. 1–6 (2013).

[2] 大野雄基，手嶋一訓，加藤大智，山岸弘幸，鈴木秀和，旭　健作，山本修身，渡邊　晃 : TLIFES を利用した徘徊行動検出方式の提案と実装，情報処理学会論文誌コンシューマ・デバイス&システム， Vol. 3, No. 3, pp. 1–10 (2013).

[3] 警察庁生活安全局生活安全企画課：平成 26 年中における行方不明者の状況 (2015). https://www.npa.go.jp/safetylife/seianki/fumei/H26yukuehumeisha.pdf

[4] 丸山敦志，旭　健作，鈴木秀和，渡邊　晃：TLIFES における低消費電力な行動判定方式の検討，平成 26 年度電気・電子・情報関係学会東海支部連合大会論文集 (2014).

[5] 出村成和：Android NDK ネイティブプログラミング，株式会社 秀和システム (2011).

[6] Java Native Interface を使用する上でのベスト・プラクティス. http://www.ibm.com/developerworks/jp/java/library/j-jni/index.html

[7] Android 端末向け消費電力測定・分析アプリ「CORE Power Profiler」 | 株式会社コア. http://www.core.co.jp/product/smartdevice/outline/corepowerprofiler.html

[8] Android KitKat |Android Developers. http://developer.android.com/intl/ja/about/versions/kitkat.html

研究業績

研究会・大会等

（1）坪井俊也，旭健作，渡邊晃：TLIFES におけるスマートフォンアプリケーションの消費電力低減の検討，平成 27 年度電気・電子・情報関係学会東海支部連合大会論文集，Sep.2015.

付 録A　GPSにかかる消費電力低減

本稿ではCPU処理を主とした構成としたため，紹介しなかった提案としてGPSにかかる消費電力低減について本付録にて示す．GPS起動時間とGPSにかかる消費電力は比例するため，3.1節のGPSによる屋外・屋内判定を見直し，GPS起動時間の削減を行った．

A.1　GPSデータの計測

GPSによる屋外・屋内判定を見直すため，GPSデータの計測を行った．

計測方法を以下に示す．

- １測定につきGPSを30秒間起動
- 計測する情報
  - 位置測位完了/不完了
  - 位置測位完了に要した時間 [ms]
  - 1秒毎に計測する情報
    - 捕捉衛星数 [機]
    - 捕捉衛星の受信強度 [dB]
- Androidアプリケーションを作成
- 使用端末
  - Samsung Galaxy Nexus （Android4.2）
  - SHARP SH-06E （Android4.2）
  - Sony SO-03F （Android4.4）

作成したAndroidアプリケーションにより，計3000回以上の計測データを収集した．計測したデータを解析して判定方式を決定する．

A.2　既存方式の評価

新たな判定方式を検討するために，まず計測データを用いて既存方式の評価を行った．

A.2.1　GPS判定の既存方式

図7にGPS判定の既存方式のフローチャートを示す．既存方式では，GPS起動10秒後に，GPS捕捉衛星数が4機未満の場合室内と判定して，測位を終了させる．4機以上の場合屋外と判定して，位置測位が完了するまでGPS起動30秒間測定を行う．

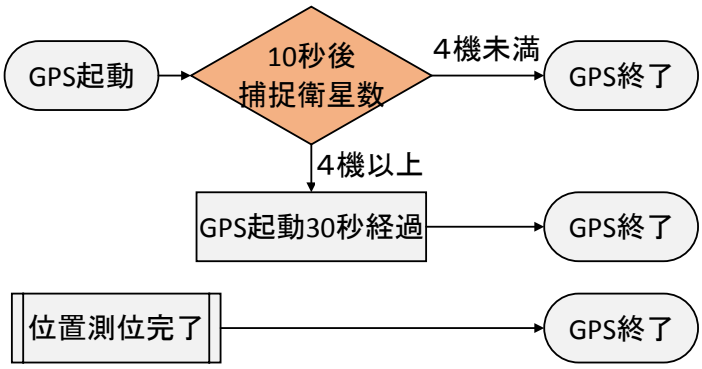


図 7　GPS 判定の既存方式のフローチャート

A.2.2　GPS 判定の評価方法

判定方式の評価には，計測したデータから求めた再現率と 1 測定あたりの平均 GPS 起動時間を用いる．また，GPS は時間とともに衛星を捕捉して，正確な位置情報を計算する．そのため位置測位ができる場所では，判定する・しないに関わらず位置測位完了するまでの時間を要する．一方，地下街等，位置測位ができない場所は，判定により無駄な起動時間を減らすことができる．そこで，実際に位置測位ができない場所の平均 GPS 起動時間も評価に用いる．

表 3 に判定結果と実際の結果の関係を示す．TP は True Positive，FP は False Positive，FN は False Negative，TN は True Negative の略である．再現率は，$\frac{TP}{TP+FN}$ により求められ，実際に取得できたもののうち，取得できると判定されたものの割合である．見守りシステムとして，位置測位の取りこぼしを少なくして，実際に位置測位できるときは測位可能と判定することを重視するため，再現率を評価に用いる．

表 3　判定結果と実際の結果の関係

		実際の結果	
		取得可	取得不可
判定結果	取得可	TP	FP
	取得不可	FN	TN

A.2.3　既存方式の評価結果

表 4 に GPS 判定既存方式の評価結果を示す．既存方式の再現率は 77.0%となっており，1 測定あたりの平均 GPS 起動時間は 9.8 秒，屋内や地下街といった位置測位ができない場所の平均 GPS 起動時間は 10.1 秒となっている．

表 4　GPS 判定既存方式の評価結果

再現率	77.0 %
1 測定あたりの平均 GPS 起動時間	9,798 ms
実際に位置測位ができない場所の平均 GPS 起動時間	10,806 ms

A.3　新 GPS 判定方式の検討

GPS 起動時間の短縮する新たな GPS 判定方式を検討する．判定方式の方針としては，既存方式と同等の再現率，かつ GPS 起動時間を短くするものとした．

図 8 に取得可否による GPS 捕捉衛星数の時間特性を示す．3000 以上あるサンプルの中において，実際に取得できた場合とできなかった場合，それぞれ GPS 起動時間 1 秒ごとの平均捕捉衛星数を求めた．

図 8 において，GPS 起動直後に捕捉衛星数が多い．これは，前回測位時の衛星情報が端末に残っているためである．そのため，前回の衛星情報が無くなった起動開始後 5 秒目から判定方式を検討することとした．

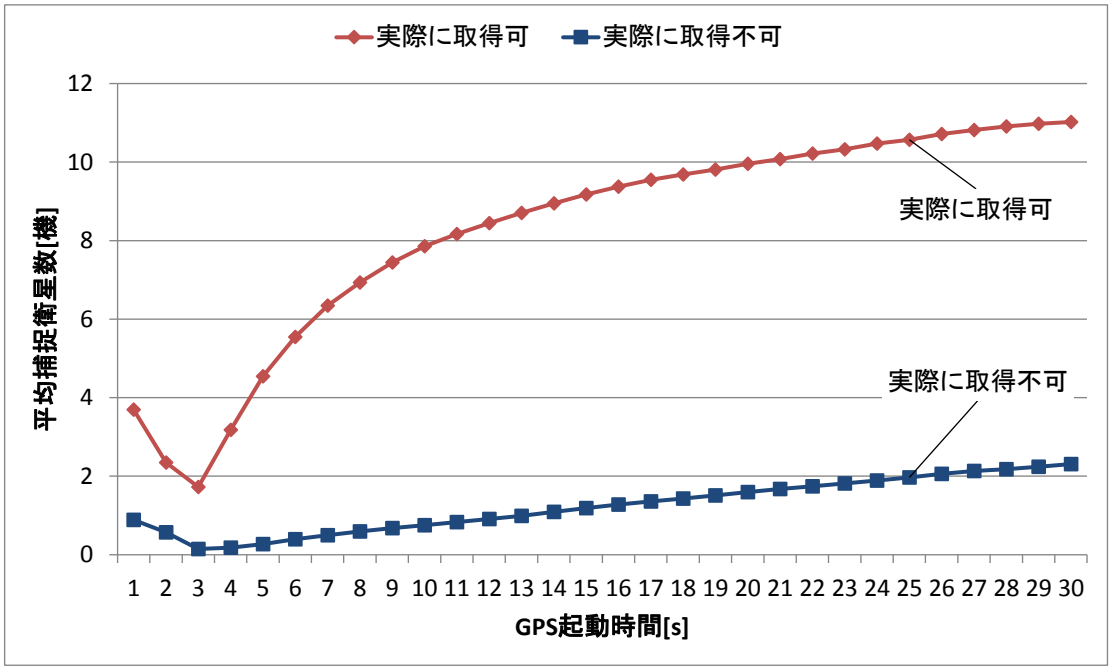


図 8　取得可否による GPS 捕捉衛星数の時間特性

表 5 に取得可能と判定する条件を 5 秒目の捕捉衛星数 > 閾値 x[機] とした場合の再現率を示す．閾値を 0 とした場合の再現率は 84.8%であり，閾値を 1 とした場合の再現率は 72.3%である．既存方式の 77.0%と同等の再現率を方針としており，捕捉衛星数では細かな閾値調整ができないため，細かな閾値を設定できる捕捉衛星の受信強度を用いて判定することとした．

表 5　取得可能と判定する条件を 5 秒目の捕捉衛星数 > 閾値 x[機] とした場合の再現率

閾値 x[機]	0	1	2	3
再現率	84.8%	72.3%	63.5%	54.7%

表 6 に取得可能と判定する条件を 5 秒目の捕捉衛星の受信強度の最大値 > 閾値 x[機] とした場合の再現率を示す．”捕捉衛星の受信強度の最大値”とは，捕捉衛星の受信強度は，捕捉した衛星ごとに取得できるため，捕捉したそれぞれの衛星受信強度の中の最大値を判定に用いる．

表 6　取得可能と判定する条件を 5 秒目の捕捉衛星の受信強度の最大値 > 閾値 $y[dB]$ とした場合の再現率

閾値 $y[dB]$	18	19	20	21	22
再現率	82.4%	80.8%	77.3%	73.5%	69.6%

表 6 中において，閾値を 20 とした場合の再現率が 77.3%であり，既存方式と同等の再現率であるため，新 GPS 判定方式を 5 秒目の捕捉衛星の受信強度の最大値 > 20$[dB]$ とした．

図 9 に GPS 判定の提案方式のフローチャートを示す．提案方式では，GPS 起動 5 秒後に，捕捉衛星の受信強度の最大値が 20dB 以下の場合取得不可の場所と判定して，測位を終了させる．20dB より大きい場合取得できる場所と判定して，位置測位が完了するまで GPS 起動 30 秒間測定を行う．

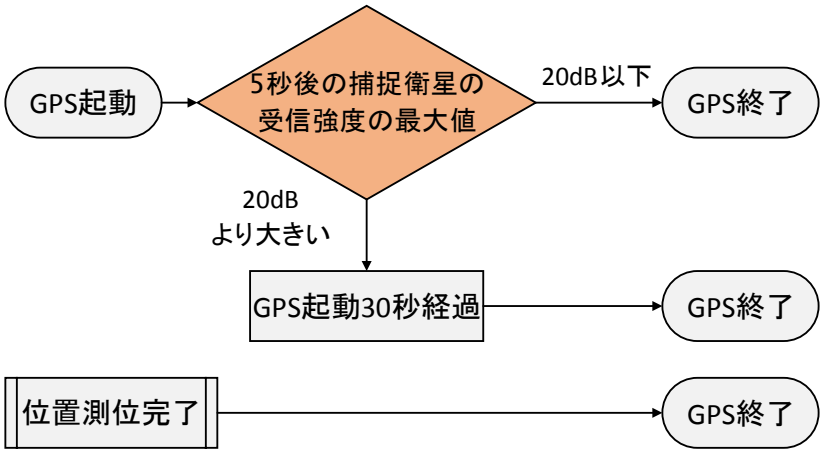


図 9　GPS 判定の提案方式のフローチャート

A.4　提案方式の評価

表 7 に既存方式と提案方式の比較評価を示す．提案方式では，既存方式と比較して 1 測定あたりの平均 GPS 起動時間を 9.8 秒から 81.7%の 8.0 秒に低減した．特に実際に位置測位ができない

場所においては，10.1 秒から 70.1%の 7.6 秒に低減した．

表 7　既存方式と提案方式の比較評価

	既存方式	提案方式	倍率
再現率	77.0%	77.3%	100.5%
1 測定あたりの平均 GPS 起動時間	9,798 ms	8,001 ms	81.7%
実際に位置測位ができない場所の平均 GPS 起動時間	10,806 ms	7,571 ms	70.1%

以上より，提案方式において，既存方式の再現率を維持したまま，屋内や地下街といった実際に位置測位ができない場所の GPS にかかる消費電力を既存方式の 70.1%に低減した．