Vol.2013-MBL-65 No.8
Vol.2013-UBI-37 No.8
2013/3/14

# Androidにおけるセンサ単位の機能仮想化

三宅 弘士[1,a)]　荒川 豊[2]　田頭 茂明[3]　福田 晃[2]

**概要**：本稿は，スマートフォンに内蔵された加速度やGPSなど数多くのセンサを，別の端末からネットワーク越しに利用可能にすることを目的として，Android上でセンサ単位の仮想化を実現する仮想化フレームワークの提案と実装を行う．さらに仮想化された別端末のセンサを利用する際の消費電力を低減するために，測定データの変化量に応じた適応的通信量制御手法を提案する．そして，仮想化された加速度センサを利用したプロトタイプアプリを作成し，機器の消費電力と擬似センサ値の精度に関して評価を行った結果を報告する．

**キーワード**：Android，センサ，仮想化

# Sensor virtualization on Android

***Abstract:*** In this paper, we propose the method to virtualize smartphone sensors on Android so that other devices can use it over Bluetooth. We also propose traffic control method according to the amount of change in the measured data in order to reduce power consumption when using another device's sensor. We prototype the application using virtualized accelerometer. And we report the results of evaluating the power consumption of the device and the accuracy of the sensor value.

***Keywords:*** Android, Sensor, Virtualization

## 1. はじめに

最近広く普及しているスマートフォン（スマホ）はさまざまなセンサが搭載されている．例えば，最新のスマホであるGalaxy S3には，8つのセンサ（加速度センサ，ジャイロセンサ，気圧計，近接センサ，デジタルコンパス，GSP/A-GPS，ライトセンサ，タッチセンサ），そしてカメラ，FMラジオ，ワンセグと，数々のセンサが搭載されている．そのためスマホを複合センサ端末としてみることもできる．一方，エアコンや冷蔵庫といった家電のIT化もめざましい．これらの情報家電は，上述したスマホのようにいくつかのセンサを搭載している．また，情報家電の中に，Android OSを組み込んだ製品も，テレビ，音楽プレーヤー，カーナビ，デジカメといった分野で広がり始めている．Android OSは，スマホの代名詞のようになっているが，本来は汎用的な組み込み用OSであり，将来的にはより多くの情報家電に組み込まれると予想される．

このように様々なセンサを搭載したスマホや情報家電の普及にしたがって，最近では単体で利用するだけでなくスマホ同士で連携したり，スマホと情報家電が連携するアプリケーションも増えてきている．例えば，スマホと連動する炊飯器や洗濯機，人感センサで自動節電するエアコン，電話がかかった時に音量を下げるTV等が登場している．また，機器をネットワークに接続するだけでシームレスに相互利用できることを目指したUniversal Plug and Play（UPnP）[*1]に関する研究も広く行われており，携帯端末で流す音楽を外出先のオーディオプレーヤーで再生する[1]など，様々なシナリオの実現が検討されている．

また，スマホに搭載されたセンサを別の端末からネットワーク越しに利用可能にする機能である，スマホをAPとして扱える機能のテザリングが備わっていることは一般

[1] 九州大学大学院システム情報科学府
Graduate School of Information Science and Electrical Engineering, Kyushu University
[2] 九州大学大学院システム情報科学研究院
Faculty of Information Science and Electrical Engineering, Kyushu University
[3] 関西大学 総合情報学部
Faculty of Informatics, Kansai University
[a)] miyake@f.ait.kyushu-u.ac.jp

[*1] http://www.upnp.org/

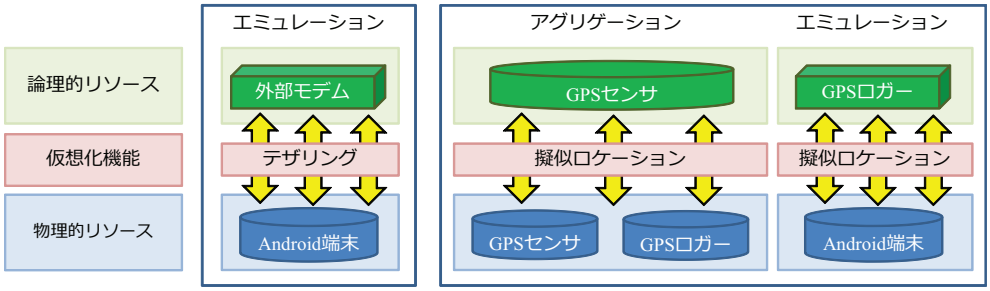


**図 1** 仮想化におけるテザリングと擬似ロケーションのパターン

的になってきている．他にも，Android 端末では GPS ロガーなどで取得した位置情報を端末全体に反映する機能の疑似ロケーションも備わっている．これらもスマホやコンピュータ同士で連携してセンサを共有するための機能である．仮想化は，コンピュータの物理的なリソースを別のタイプの論理的なリソースとして扱うエミュレーション，コンピュータの複数の論理的なリソースを一つの論理的リソースとして扱うアグリゲーション，コンピュータの一つの物理的なリソースを複数の論理的リソースとして扱うパーティショニングから成り立っている．現行の Android で実現されている代表的な仮想化機能はテザリングと擬似ロケーションであり，それぞれエミュレーションとアグリゲーションである．

Android におけるこれらの仮想化として，これらの無線通信デバイスと GPS センサだけでなく，他のセンサにおいても外部端末から利用できる仮想化機能を実現することで，様々なシステムに適用できると考え本研究を行った．また，本研究により Android におけるセンサ単位の仮想化が可能となれば，スマホどうしで機能を補完することだけでなく，Android を搭載した家電製品と連携したシステムやサービスも可能となるため，学術的にも産業的にも意義深い研究である．本研究は，現時点ではきわめて斬新であるが，テザリング機能が一般的に使われているように，センサ単位での端末間連携も将来的に一般的な機能となると考える．センサ単位の仮想化によって実現できる機能の例を以下に挙げる．

- Android 端末が拾った音声を別の端末で保存
- iPhone で撮った写真を Android の SD カードに保存
- 高精度な車載 GPS をスマホで利用して周辺情報を検索
- エアコンの温度計をスマホで利用
- Android で取得した天気情報とエアコン設定の連携

そこで，本稿では，これらを実現するための仕組みについて検討し，Android を用いてプロトタイプの実装を行った．提案手法では，通常の Android における GPS の仮想化手法（擬似ロケーション機能）の設計アーキテクチャを参考に，SensorManager の独自拡張を行った．これにより，センサー情報を仮想化したとしても，他のアプリケーションは何も変更する必要がないアーキテクチャとなっている．しかしながら，OS そのものを改変したため，市販されている携帯端末のままでは利用できないものである．将来的には，擬似ロケーションと同様に，OS 内に標準的に組み込まれるようになることを狙っている．

以降，第 2 章では関連技術に関して説明する．また，第 3 章で提案手法について説明し，第 4 章ではプロトタイプの実装について説明する．そして，第 5 章で提案手法の評価について説明し，最後に，第 6 章で本研究のまとめを述べる．

## 2. 関連技術

現在の Android で実現済みの仮想化として，通信の仮想化であるテザリング，位置情報の仮想化である擬似ロケーションについて説明する．また，スマホのセンサを共有する研究としてスマホ仮想化に関する研究を説明する．

### 2.1 テザリング

テザリングは，スマホのようにデータ通信機能を備えたコンピュータを外付けモデムとして用いて，他のコンピュータでデータ通信をする機能である．Android では Wi-Fi テザリング，USB テザリング，Bluetooth テザリングを利用できる．USB テザリングでは，Android 端末と USB ケーブルで物理的に接続することで，他のコンピュータでデータ通信を利用できる．Wi-Fi テザリングでは，Android 端末が Wi-Fi のアクセスポイント（AP）としての役割を担い，他のスマホや PC，ゲーム機などで Wi-Fi を通して接続してデータ通信をすることができる．Bluetooth テザリングでは，Wi-Fi テザリングと同様に物理的な接続を必要とせず，他のコンピュータから Bluetooth を通してデータ通信を利用できる．したがって，仮想化の観点からテザリング機能について考えると，この機能は Android 端末を他のコンピュータから外部モデムとして利用しており，仮想化パターンのエミュレーションであるといえる（**図 1**）．

文献 [2] では，スマホの Wi-Fi テザリング利用時に，Wi-Fi インターフェースを適切なタイミングでスリープさせる手法とそのスケジューリングアルゴリズムを提案している．これにより Wi-Fi テザリング使用時の消費電力削減を実現している．

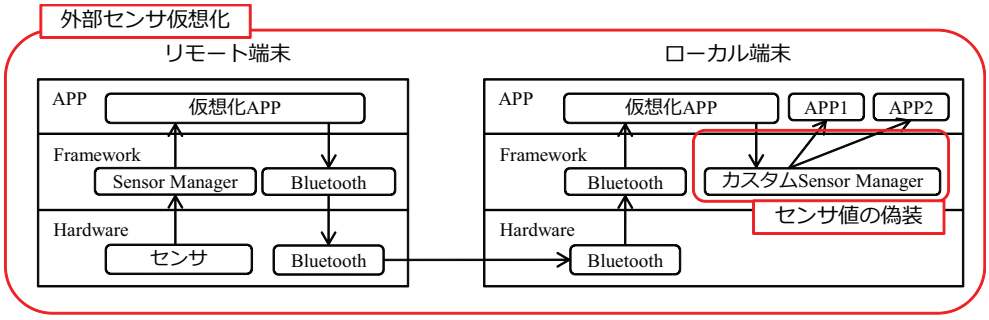


**図 2** 外部センサ仮想化の概要

### 2.2 擬似ロケーション

Android では，バージョン 1.5 以降では擬似ロケーション機能を利用することができる．擬似ロケーション機能を利用することで，Android 端末において自身の GPS センサで取得した位置情報を利用せずに，アプリケーションから入力する任意の位置情報を端末全体に反映させることができる．したがって，地域限定の情報を提供したり特定の場所に行くことでイベントが起きるような GPS の位置情報を利用した Android 用アプリケーションにおいて，実際にユーザが現地に赴くことなく，あたかもその場所にいるかのように振る舞うこともできる．また，この機能の有効な利用方法として，高精度の外部の GPS ロガーなどを利用する方法が挙げられる．GPS ロガーと Android 端末を Bluetooth によって通信させることで，GPS ロガーで測定した位置情報を Android 端末自身の測定として利用できる．GPS ロガーと連携するためのアプリケーションは，Google Play Strore[*2] に複数登録されている．例えば，‘‘Bluetooth GPS[*3]’’ は擬似ロケーション機能を利用して，外部の GPS ロガーで取得した位置情報を Android 端末で利用するアプリで，‘‘Bluetooth GPS Output[*4]’’ は Android 端末を GPS ロガーとして，他のコンピュータに位置情報を提供するアプリである．

ここで，擬似ロケーション機能において二つの仮想化が行われていると考えることができる．一つは，端末自身に搭載されている GPS センサと外部 GPS ロガーを一つの論理的 GPS センサとして扱えるようにするアグリゲーション，もう一つは，Android 端末を他のコンピュータから GPS ロガーとして扱えるようにするエミュレーションである（図 1）．エミュレーションの利用例として，Android 端末を複数持っているユーザや，iPhone を GPS ロガーとする場合が考えられる．これは，スマホの機種によって，利用されている GPS センサの精度はまちまちであり，複数のスマホを持ち歩くユーザにとって，どの端末においてもより良い位置情報を利用したいといった要望があるためである．

### 2.3 スマートフォン仮想化

文献 [4] では，仮想化スマホを作成し，それを IP を通してスマホから利用するシステムを提案している．このシステムでは，仮想化された論理的なスマホはセンサを持っていないため，クライアントである物理的なスマートフォンの各センサで取得したセンサ値を 3G などの IP によってサーバである仮想化スマホに送る．これにより，仮想化スマホにおいて，外部の物理的なスマホのセンサの利用を実現している．

## 3. 提案手法

現在普及しているスマートフォンは，複数の無線ネットワークを利用できるだけでなく，加速度センサ・ジャイロセンサ・輝度センサなどの多様なセンサも搭載している．本研究の目的は，データ通信機能を共有するテザリングと同じように，各センサを端末間で共有し，機能を相互補完可能にする仕組みを創出することである．したがって，ここでは Android 上でセンサを仮想化する手法を提案する．このセンサ仮想化を実現するにあたり，大きく分けて二つの課題が挙げられる（**図 2**）．一つは，Android OS 上でセンサ値を偽装する仕組みをつくることであり，もう一つは，リモート端末とローカル端末間でセンサを共有する仕組みをつくることである．ここで，リモート端末はセンサ値を送信する端末，ローカル端末はセンサ値を受信し，それを偽装センサ値として使用する端末を指す．以下では，これらの二つの仕組みについて説明する．

### 3.1 センサ値を偽装する仕組み

Android では，外部端末のデータ通信機能を共有する “テザリング機能” や，緯度・経度を手入力することでそれを自前の GPS センサで取得したセンサ値として扱う “疑似ロケーション機能” を利用できる．しかしながら，Android 端末はその他にも加速度センサ・ジャイロセンサ・輝度センサ・近接センサ・気圧センサなどを搭載しているが，こ

*2 https://play.google.com/store
*3 https://play.google.com/store/apps/details?id=googoo.android.btgps&feature=search_result#?t=W251bGwsMSwyLDEsImdvb2dvby5hbmRyb2lkLmJ0Z3BzIl0.
*4 https://play.google.com/store/apps/details?id=com.meowsbox.btgps&feature=search_result&hl =ja

れらのセンサを仮想化する仕組みは備わっていない．そのため，Android において複数端末間でセンサを共有するためには，疑似ロケーション機能のように，手入力によってそれぞれのセンサ値を入力する機能の実現が不可欠である．この機能の追加先として，アプリケーションやユーザレベル，OS の中，または Java VM などの仮想化層が選択肢となるが [3]，既存の Android では，アプリケーションを作成するだけでセンサを仮想化できないため，OS そのものに変更を加えるアプローチをとる．これは Android はオープンソース・オープンプラットフォームであり，OS 全体のソースコードやツールが公開されているため，OS に変更を加えてカスタム ROM をつくることができるからである．

図 2 の “センサ値の偽装” と囲った部分が簡単な Android アーキテクチャにおける変更を加える部分である．アプリケーションフレームワークのうち，センサの制御を行う SensorManager などを変更することでセンサ仮想化を実現する．

### 3.2 センサ仮想化の仕組み

Android において手入力によって，任意のセンサ値を端末全体に反映できる場合，センサを仮想化するために次に必要となるのは，複数の Android 端末間でセンサを共有する仕組みである（図 2）．二つの端末間でのセンサ仮想化の概要を以下に説明する．

- リモート端末
  自前のセンサで取得したセンサ値を利用する端末．この端末はセンサの仮想化が可能である必要はない．
- ローカル端末
  リモート端末から受信したセンサデータを自分のセンサ値として利用する端末．この端末はセンサの仮想化が可能である必要がある．
- 通信方法
  Bluetooth を利用してリモート端とローカル端末間で通信を行う．Wi-Fi や 3G を使うこともできるが，仮想化の利用環境として二つの端末が近くにある場合を想定しているため，通信による消費電力が低い Bluetooth を使用する．
- 通信データ
  仮想化するセンサのセンサ値（public final float[] values），タイムスタンプ（public long timestamp）を byte 形式で送受信する．

通信方式として Bluetooth を採用しており，3G や Wi-Fi など比較すると低消費電力ではあるが，Bluetooth の通信による消費電力は通信回数に依存するため，リモート端末がセンサ値を取得するたびにローカル端末に送信することは，バッテリー容量の制限が厳しい携帯端末においては必ずしも良い方法とは言えない．一方で，送信間隔を伸ばすことで送信回数を減らすだけでは，誤差の面においてセンサ値を利用する上位のアプリケーションの要求に答えられない．したがって，この問題を解決するためには，通信回数を減らすことで消費電力を減らしつつ，アプリケーションの要求する精度を満たすための通信手法を考える必要がある．ここではわれわれの提案する通信プロトコルについて説明する．

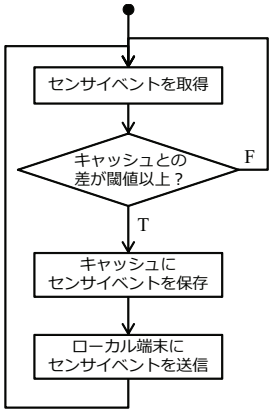


**図 3** 提案手法におけるリモート端末のアルゴリズム

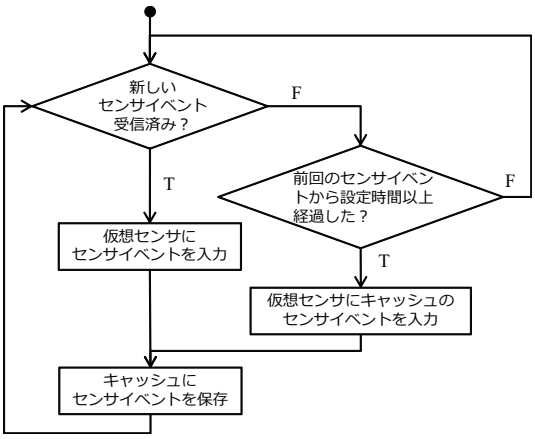


**図 4** 提案手法におけるローカル端末のアルゴリズム

提案手法では，リモート端末で取得したセンサ値と前回取得したセンサ値の差分に閾値を設定し，差分が閾値以上の場合はローカル端末にセンサデータを送信し，閾値以下の場合は送信しないこととする．このとき，ローカル端末では一定時間以上リモート端末からセンサ値が送られてこない場合，前回のセンサ値からの変化が閾値以下であると判断し，キャッシュに残していた前回のセンサ値を使用する．これにより，二端末間での通信回数を減らしつつ，センサ値のサンプリング数を維持することができる．

提案手法におけるリモート端末とローカル端末におけるアルゴリズムについて，それぞれ**図 3**, **図 4** を参照して説明する．まず，リモート端末では，センサ値を取得するとキャッシュデータのセンサ値との差分を計算する．この差分が閾値よりも大きい場合，取得したセンサイベントをキャッシュに上書きし，ローカル端末にセンサイベントを

送信する．差分が閾値以下の場合は取得したセンサイベントを破棄して次のセンサイベントの取得まで処理は行わない．

次に，ローカル端末では，まずリモート端末からセンサイベントが送られてきているか確認し，送られてきているときはこのセンサイベントをキャッシュに保存し，仮想化したセンサのセンサイベントとして端末全体に反映する．送られてきていない場合は，前回データを受信してから経過している時間とサンプリング間隔の閾値を比較し，この閾値以上経過している場合はローカル端末のキャッシュに保存していたセンサイベントを仮想化したセンサのセンサイベントとして使用する．経過した時間が閾値以下の場合は，再びセンサイベントが送られているか確認する．以上が提案手法のアルゴリズムである．

## 4. 実装

ここでは作成した提案手法のプロトタイプについて説明する．

### 4.1 API

Android においてアプリケーションからセンサを制御するには SensorManager 利用する．したがって，アプリケーションからセンサ値を手入力で設定できるようにするために，Android OS と SDK を再コンパイルして以下の API を追加した．

- boolean registerMockSensorListener(Sensor sensor, int rate)
  指定したセンサを指定したサンプリング間隔で仮想センサとして登録する．これにより，setMockSensorEvent によって入力された値をセンサ値として利用できる．サンプリング間隔は，間隔の短い順に，SENSOR_DELAY_FASTEST(FASTEST)，SENSOR_DELAY_GAME(GAME)，SENSOR_DELAY_UI(UI)，SENSOR_DELAY_NORMAL(NORMAL) の四種類がある．
- boolean unregisterMockSensorEventListener(Sensor sensor)
  指定したセンサを仮想センサから解除する．
- void setMockSensorEvent(Sensor sensor, float[] values, long timestamp)
  指定した仮想センサのセンサ値とそのタイムスタンプを入力する．registerMockSensorListener によって擬似センサに登録していない場合はこのセンサ値は反映されない．

これにより，Android においてアプリケーションから，仮想化するセンサとそのセンサのサンプリング間隔を指定して擬似センサとして登録することで，任意のセンサ値を端末全体に反映させることができる．

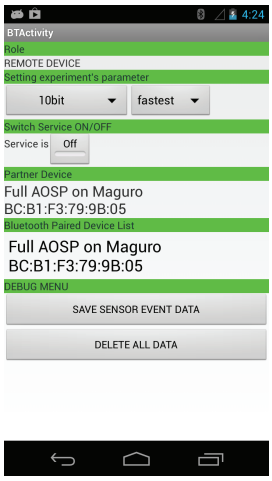


**図 5** リモート端末用仮想化アプリの画面

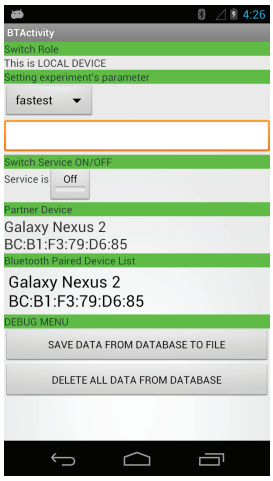


**図 6** ローカル端末用仮想化アプリの画面

### 4.2 仮想化アプリケーション

提案手法のプロトタイプとして，加速度センサを仮想化するためのリモート端末とローカル端末用アプリケーションを実装した．これらのアプリケーションにおいて設定するパラメータを説明する．

まず，リモート端末（**図 5**）では，センサ値差分に対する閾値を 10bit〜5bit，または閾値なしから選択する．また，サンプリング間隔を FASTEST，GAME，UI の 3 つから選ぶ．さらに，Bluetooth によって通信するローカル端末を選択する．

次に，ローカル端末（**図 6**）では，待機時間を FASTEST，GAME，UI の 3 つから選ぶ．また，新しいセンサイベントを受信済みか確人するポーリング間隔を調整する場合は，ms 単位で数値を入力し，設定する．そして，Bluetooth によって通信するリモート端末を選択する．

以上のパラメータを設定したうえで二端末のサービスを実行することで加速度センサの仮想化を行うことでできる．

## 5. 評価

ここでは，提案した Android OS 上でセンサを仮想化する仕組みと，別端末のセンサで取得したセンサ値をローカル端末のセンサ値として偽装する仕組みを実装し，それぞれのサンプリング間隔において提案手法を適応した精度と消費電力の評価を行う．今回は特に，Android に搭載されているセンサの中でも加速度センサについて仮想化を行い，提案手法を適応した．さらに，リモート端末とローカル端末におけるそれぞれのセンサ値を解析することにより提案手法を評価する．

### 5.1 実験方法

ここでは，各サンプリング間隔に対して従来手法と提案手法を適用した際のセンサ値の誤差とセンサ値送信回数を測定し，解析する方法を説明する．従来手法では，リモー

ト端末において取得したセンサ値を全て送信し，ローカル端末において，Bluetooth の受信をブロッキング処理で行うものとする．

### 5.1.1 実験環境

提案手法の定量的評価実験を行う際の実験環境を以下に示す．

- 使用する端末
  リモート端末，ローカル端末ともに Galaxy Nexus を使用
- 対象のセンサ
  本実験では加速度センサを対象とする．
- サンプリング間隔
  リモート端末がセンサイベントを取得する間隔．Galaxy Nexus の加速度センサでは UI と NORMAL は同じサンプリング間隔であるため，本実験では FASTEST，GAME，UI の三つに対して評価する．ここで，GAME，UI，NORMAL の Android OS のソースコード上でのサンプリング間隔はそれぞれ 20ms，67ms，200ms である．しかしながら，このサンプリング間隔は目安であり，この通りの間隔で必ず届くわけではない．通常，ソースコード上の値よりも早く届き，FASTEST では可能な限り早くセンサ値を取得する．
- センサ値の閾値
  リモート端末からローカル端末にセンサ値を送信するかどうか判断する際のセンサ値差分の閾値．各センサのセンサ値の最大値と解像度から計算する．本実験で使用する Galaxy Nexus では，加速度センサの最大値 19.6133[m/$s^2$]，解像度 0.038344003[m/$s^2$] である．これにより，加速度センサのセンサ値は-19.6133〜19.6133 の値をとりうるため，約 10bit の精度であるといえる．したがって，これを指標として，精度が 5〜10bit になるように閾値を設定する．表 1 に加速センサにおけるセンサ値の精度と閾値の対応を表す．
- リモート端末における待機時間
  ローカル端末においてキャッシュのセンサ値を使用するまでの待機時間．これには設定したサンプリング間隔における Android OS ソースコード上の数値を採用する．FASTEST ではこれが 0ms であるため，各端末に搭載されている各センサでのセンサ値取得最短間隔を代わりに採用する．Galaxy Nexus の加速度センサでは 10ms である．
- 測定中の動作
  加速度センサを用いて，雪山を滑り降りるペンギンを操作するレーシングゲームである‘‘Penguin Skiing 3D[*5]’’ の Bunny Hills コースをプレイする．このゲームでは，加速度の値によって，スピードと左右のコントロールを行っている．
- 端末間の距離
  リモート端末とローカル端末の距離は常に 1m 以下に配置する．
- 端末の設定
  各端末の Wi-Fi の電源を OFF にして実験を行った．

**表 1** 加速度センサにおける精度と閾値の対応

| センサ値精度 [bit] | 閾値 [m/$s^2$] |
|---|---|
| 10 | 0.038344003 |
| 9 | 0.076688006 |
| 8 | 0.153376012 |
| 7 | 0.306752024 |
| 6 | 0.613504048 |
| 5 | 1.227008096 |

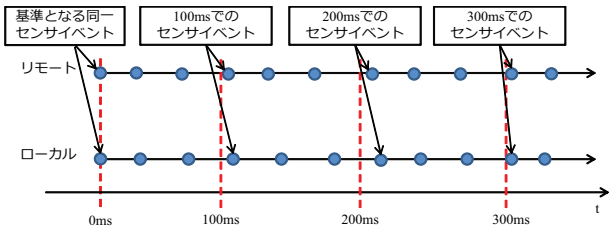


**図 7** センサ値誤差の計算方法

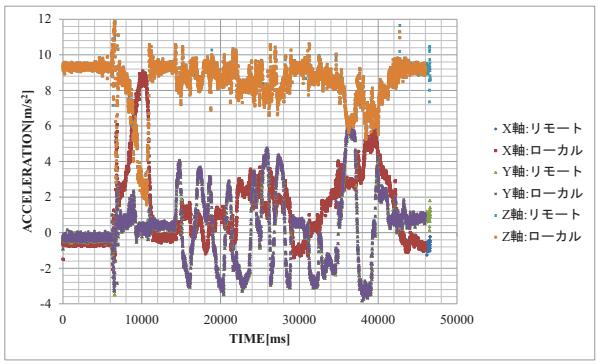


**図 8** サンプリング間隔 FASTEST 従来手法における加速度の時間変化

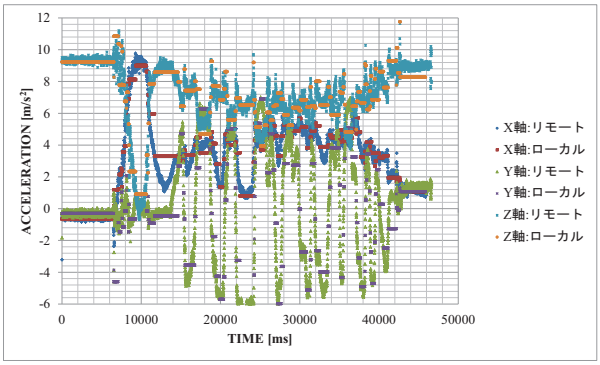


**図 9** サンプリング間隔 FASTEST 精度 5bit における加速度の時間変化

- 測定中の動作
  加速度センサを用いて，雪山を滑り降りるペンギンを操作するレーシングゲームである‘‘Penguin Skiing 3D[*5]’’ の Bunny Hills コースをプレイする．このゲームでは，加速度の値によって，スピードと左右のコントロールを行っている．
- 端末間の距離
  リモート端末とローカル端末の距離は常に 1m 以下に配置する．
- 端末の設定
  各端末の Wi-Fi の電源を OFF にして実験を行った．

[*5] https://play.google.com/store/apps/details?id=com.canadadroid.penguinskiing/

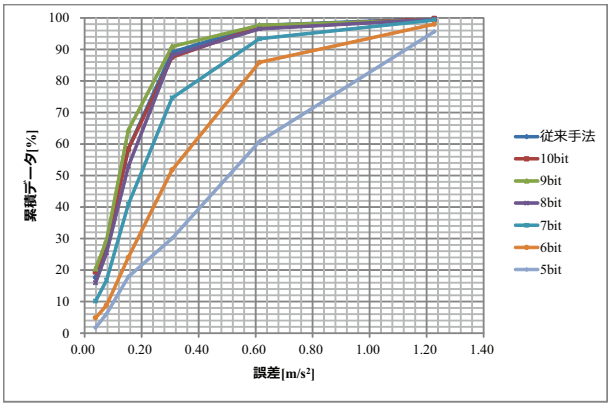


**図 10** サンプリング間隔 FASTEST における各精度の誤差

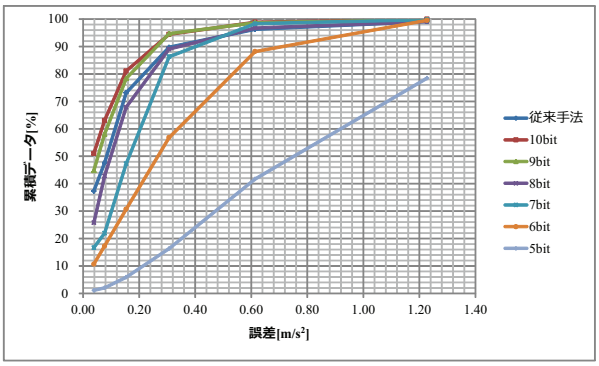


**図 11** サンプリング間隔 GAME における各精度の誤差

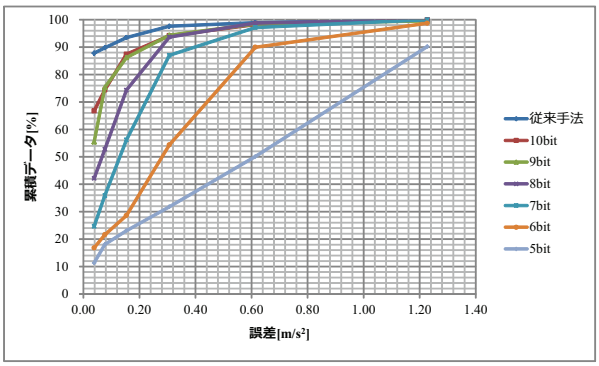


**図 12** サンプリング間隔 UI における各精度の誤差

### 5.2 解析方法

二端末における加速度の時間変化をグラフで表し，波のずれを見るだけでなく，二端末間のセンサ値誤差，送信率を計算することで提案手法を評価する．センサ値誤差は，二端末において 100ms 間隔でセンサ値を取得し（**図 7**），それぞれの端末における x，y，z 軸方向のセンサ値の差の絶対値を取り，足して 3 で割ったものとする．送信率は，リモート端末からローカル端末へのセンサ値送信回数をリモート端末でのセンサ値取得数で割ったものとする．

#### 5.2.1 実験結果

サンプリング間隔 FASTEST における，従来手法と精度 5bit でのリモート端末とローカル端末の加速度の時間変化を比較したグラフをそれぞれ**図 8**，**図 9** に示す．次に，**図 10**，**図 11**，**図 12** にそれぞれサンプリング間隔 FASTEST，GAME，UI における各精度での誤差を示す．そして，**表 2** に各サンプリング間隔における各精度での送信率をまとめて示す．

**表 2** 各サンプリング間隔における各精度での送信率

| サンプリング間隔 | 従来手法 | 10bit | 9bit | 8bit | 7bit | 6bit | 5bit |
|---|---|---|---|---|---|---|---|
| FASTEST | 100 | 96 | 82 | 39 | 16 | 6 | 2 |
| GAME | 100 | 93 | 75 | 41 | 19 | 8 | 3 |
| UI | 100 | 85 | 67 | 44 | 26 | 13 | 6 |

#### 5.2.2 考察

上記の定量的評価実験の結果を踏まえ提案手法について考察する．

#### 5.2.3 サンプリング間隔 FASTEST

まず，サンプリング間隔が最も短い FASTEST の場合，図 8 より従来手法では，二端末間の加速度の波はほぼ一致しており，図 9 より提案手法で精度をさげると，ローカル端末の加速度がとびとびの値をとるようになり，二端末間の加速度の波が大きくずれることがわかる．したがって，誤差が従来手法よりも小さい精度を利用することで上位のアプリを利用するユーザにとってパフォーマンスが良いといえる．図 10 より提案手法の精度 10bit,9bit において従来手法よりも誤差が小さいことがわかる．表 2 より，この中で精度 9bit が最も送信率が低く，従来手法と比べて 82% まで削減できる．したがって，サンプリング間隔 FASTEST において，誤差が抑えつつ送信率を下げることができる精度 9bit が最適であるといえる．

#### 5.2.4 サンプリング間隔 GAME

次に，サンプリング間隔が GAME の場合，図 11 より提案手法の精度 10bit，9bit において従来手法よりも誤差が小さいことがわかる．表 2 より，この中で精度 9bit が最も送信率が低く，従来手法と比べて 75% まで削減できる．したがって，サンプリング間隔 GAME において，誤差が抑えつつ送信率を下げることができる精度 9bit が最適であるといえる．

#### 5.2.5 サンプリング間隔 UI

サンプリング間隔が UI の場合，図 12 より提案手法おいて従来手法よりも誤差が小さい精度はないことがわかる．したがって，サンプリング間隔 UI において，誤差が最小である従来手法が最適であるといえる．従来手法を使用する場合，送信率を下げることはできないが，FASTEST，GAME，UI において送信率 100% での送信回数はそれぞれ約 5800 回，2900 回，720 回であり，UI における基本的な送信回数自体が他二つのサンプリング間隔と比べて少なく，送信率を下げるメリットも低いといえる．

## 6. おわりに

### 6.1 結論

外部センサ仮想化では，Android において SensorMan-

ager クラスで扱えるセンサにおいてセンサ値を偽装する手法を提案した．さらに，Android においてセンサを仮想化する仕組みを提案した．そして，センサ仮想化時の通信方法として，センサ値の変化量に閾値を設定することでセンサ値の精度を保ちつつ通信による消費電力を削減する手法を提案した．結果として，サンプリング間隔 FASTEST とサンプリング間隔 GAME では精度 9bit，サンプリング間隔 UI では従来手法が最適であり，全てのセンサ値を送受信する従来手法と比べて，FASTEST と GAME ではそれぞれ 82%，75% に送信回数を削減できることを確認した．

### 6.2 今後の展望

外部仮想化において本論文では，Android に搭載されているセンサの中でも加速度センサに対して提案手法を適用した．そのため，今後は，ジャイロセンサや近接センサなどのセンサ仮想化ができるように実装を行い，それらを利用したシステムやサービスを検討する．

また，本研究の提案手法では，Android の SensorManager クラスで扱えるセンサを仮想化する手法を提案したが，カメラやマイクなど別のクラスで扱われるセンサに対しても仮想化を行う手法を検討する．

### 参考文献

[1] Fasbender, A. Gerdes, M. Matsumura, T. Haber, A. and Reichert, F. : *Media Delivery to Remote Renderers Controlled by the Mobile Phone*, 6th IEEE Consumer Communications and Networking Conference, pp. 1-2, Jan. 2009.

[2] Hao H., Yunxin L., Guobin S., Yongguang Z. and Qun L: *DozyAP: Power-Efficient Wi-Fi Tethering*, Proc. of the 10th international conference on Mobile systems, applications, and services, Jun. 2012.

[3] Rudolph, L: *A Virtualization Infrastructure that Supports Pervasive Computing*, IEEE Pervasive Computing, Vol. 8, no. 4, pp. 8-13, Oct.-Dec. 2009.

[4] Eric Y. C. and Mistutaka I.: *Virtual Smartphone over IP*, 2010 IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks, pp. 1-6, Jun. 2010.