IoTセンサーに対するソフトウェアからの死活監視

杉本 一彦[1,a)]　串田 高幸[1]

概要：2020 年に入り 5G が本格的に普及し始め，IoT 分野は急速に成長していくと様々なアナリストが分析をしている．これに伴って IoT(Internet of Things) センサー，つまりワイヤレスセンサーネットワークにつながれているセンサーの個数は限りなく増加するものであると考えられる．膨大な個数のセンサーを正確に監視し，問題が起きた事実を適切にユーザーあるいはエンジニアに伝えることが非常に重要である．しかし，センサーの個数に伴いその種類も多種多様になってきており，一度で全てを監視することは非常に困難であるとされている．このことにより，現在，センサーに対する監視は未成熟であり，研究論文として公開されているものも数少ない．センサーに問題が起きた際，人が直接センサーを確認することなく問題の把握さらに問題の把握が行えたと仮定するならば，センサーに対する信頼性は極めて大きく担保されることになるであろう．これらの課題を踏まえ本実験ではセンサーの監視の初期段階として I^2C 通信プロトコルを用いて接続されるセンサーの死活監視を行った．その結果，センサーに電源が繋がっているか否かだけではなく，電源に繋がっていてかつデータが送信できる状態にあるか否かを判別することができた．本稿では，その提案方法，実装，評価を記す．

1. はじめに

インターネットは劇的に進化し，様々な接続方法が作成されている．モノのインターネット (Internet of Things : IoT) は現在のインターネット通信を Machine-to-Machine(M2M) に変換する方式の一種である．そのため，モノのインターネット (IoT) は様々な種類の高機能なセンサーを組み込んだ現実世界とサイバースペースを途切れることなく接続することが可能である．インターネットに接続された多数のマシンは，日常生活をより便利にし，有益なサービスを提供するのに役立つ膨大な量のデータを生成および交換する [1].

そしてその膨大な量のデータを生成しているのはワイヤレスネットワークに接続されている多数のセンサーである．そのセンサーの数は年々増加しており，総務省平成 30 年版情報通信白書によると 2020 年には約 400 億台になると予想され，そのデータ量は年間約 80ZB とも言われている [2]．特に，今後はコネクティッドカーの普及により IoT 化の進展が見込まれる「自動車・輸送機器」，デジタルヘルスケアの市場が拡大している「医療」，スマート工場やスマートシティが拡大する「産業用途 (工場，インフラ，物流)」などの分野が急激に成長すると予想され，IoT センサーの数は更に増加していくと考えられている [3].

[1] 東京工科大学コンピュータサイエンス学部
CDSL, TUT, Hachiouji, Tokyo 101–0062, Japan
[a)] C0117164

このような状況の中で IoT センサーを全て一括で管理することは極めて重要な課題である．重要な分野や産業に普及するにつれ，IoT の管理は元よりその大本であるセンサーが巻き起こす故障などのリスクは極めて大きな影響力を持つことになってくる．センサーがどのような状態であるかを監視することはリスクを回避するためにも重要である．しかし現状はそうではない．ビックデータの処理では数多くの研究者が数多くの方式を考えているが，センサー自体の監視やセキュリティに関する問題はまだ課題が山積みである [4][5][6]．本稿では，センサーの死活監視を行うソフトウェアの提案をし，センサーに何らかの異常が発生しデータが取得できないくなった場合即座にユーザへアラートを出すようなシステムを開発，また評価を行う．

本稿は以下このように構成されている．第 2 章：関連研究について取り上げ本研究と比較を行う．第 3 章：センサーの死活監視の課題について述べる．第 4 章：第 3 章で取り上げた課題に対する解決策の提示を行う．第 5 章：第 4 章で述べた解決策の実装について述べる．第 6 章：第 5 章で実装したシステムの評価を行う．第 7 章：第 3 章から第 6 章までの考察を行う．第 8 章：本稿のまとめを行う．

2. 関連研究

概要にて先述したように，センサーの監視に関する研究論文はほとんど存在しない．まずは IoT におけるセンサーとはどのようなものかを紹介していく．

IoT センサーとは一般のセンサーとは違い，IoT ゲートウェイと呼ばれるマイクロコンピュータ，マイクロサーバにセンサーを接続する必要がある [7]．以下に画像を引用する．これは IoT ゲートウェイとセンサーを有線接続と無線接続で繋いだ場合の構成図をそれぞれ示している．有線で接続することでタイムラグや回線が混在する状況を防ぐことができ用意に管理ができ，無線では場所を特定することなく，接続することが可能である．マイクロコンピュータやマイクロサーバに代表されるのは RaspberryPi である [8]．また，マイクロコンピュータとマイクロサーバの違いは OS がインストールされているか否かということだけである．マイクロコンピュータは OS がインストールされていないため構造がシンプルでかつ処理能力も小さくできコストが抑えられるメリットがあり，マイクロサーバは OS が入っているため行える処理が非常に多くアプリケーションソフトウェアの柔軟性が向上するメリットの反面，構造が複雑化し，処理能力も求められるためコストが大きくなってしまうというデメリットがある [9]．本研究のセンサーの監視は IoT ゲートウェイのソフトウェアから行うため，マイクロサーバである必要がある．また最近ではスマートフォンをゲートウェイにするという仕組みもあるようだ [10]．スマートフォンを IoT ゲートウェイとすることで，ウェアラブルセンサーを IoT として取り込める可能性も出てくるなど，IoT の幅はさらに広がるであろう．またデバイスの接続の開始や切断を制御する信号の増加により，ゲートウェイの負荷増大や輻輳発生を解決する手段の一つにもなる [11]．IoT ゲートウェイにて監視のソフトウェアを走られるため，本研究を発展させ，スマートフォンにも適応可能なシステムの構築が求められている．

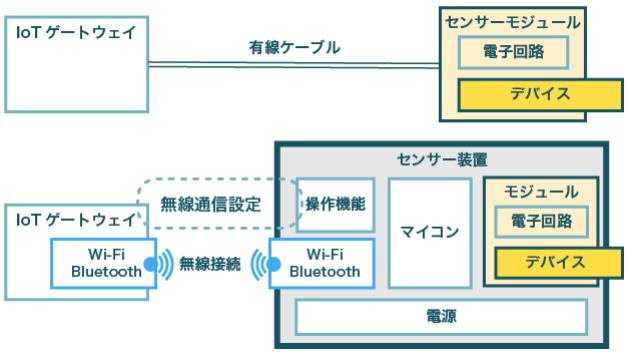

図 1　IoT ゲートウェイとセンサー

3.　課題

センサーはワイヤレスネットワーク接続ではなく RaspberryPi などをゲートウェイとして直接回路で繋がれており，以下の図 1 の様にネットワークを介したセンサー自体の状態情報を取得することはできない．[12]　そのためセンサーの状態情報を取得するためにはインターネットとの接続口である RaspberryPi などをゲートウェイと見立てることができるデバイスを用いて行わなければならない．しかし回路に繋げていて電源に繋がっている状態であってもデータが取れなければそれはセンサーが正常に動作しているとは言い難い．センサーが正常に動作している，とは，回路に適切に接続されており，かつデータが適切に取得できる状態のことを指す．
しかし，これを基準としてセンサーの死活を監視することは容易なことではない．その理由として考えられることを以下に記す．

(1) V_{dd}（電圧）の値では，センサーが回路に適切に接続されているか否かは判別できるが，データの取得が準備できているか否かは判別できない．
(2) センサーに設定されているレジスタ値はセンサー固有のため，設定を統一することができない．
(3) センサーからデータを取得するためには，センサーとどの通信プロトコルを用いて接続するのかを事前に把握しなくてはならない．そしてその通信プロトコル固有の接続確認方法を取らなくてはならない．(I^2C 通信プロトコルでは”i2cdetect”というコマンドを利用してセンサーが正常に繋がっているかを判別することができる．)

また，センサーから任意のデータを取得するためにはそのセンサーのデータシートに記述されているレジスタ値や換算式を用いてセットアップをすることでようやくデータを取得することができる．このセットアップはハードウェアの知識が必要になる事もあり，一般にソフトウェアのみを学習しているエンジニアには向かない．また，センサーごとにそのセットアップ方法が異なるため複数個の異なるセンサーがある場合時間がかかってしまう．センサーの取り付けが完了し，データが取得できる準備が整い次第，すぐに目的のデータを取れることが非常に望ましいものと考えられる．

4.　解決策の提案

センサーの死活監視は IoT では極めて重要な課題である．しかし 3 で述べた通り，全てのセンサーに対して一括で死活監視することは難しいため，センサーの種類を I^2C 通信プロトコルを用いるものだけを対象とし死活監視を行う．

4.1 IoT ゲートウェイ

センサーを監視するためにはまず IoT ゲートウェイはマイクロサーバである必要がある [9]．OS が入っていることにより，ソフトウェアが柔軟に開発でき，また本研究で使用するコマンド (i2cdetect) は Linux 系 OS でしか使えないものになっているからである．i2cdetect コマンドについては後述する．

4.2 i2cdetect の利用

3 でも述べたが，電源に繋がっている状態を取得することだけではセンサーが正常に動作しているかは不明である．電源に繋がっているのにも関わらず，データが適切に送信されないことやそもそもセンサーが繋がっているという判別さえされないことがあり，V_{dd} の値だけではやはり監視には不十分である．ではどのようにしてセンサーからデータが送られる状態になっているかを判別するのか．それを解決するのが i2cdetect というコマンドなのである [13]．i2cdetect は，Linux からセンサーに対してのアクセスコマンドである．これは，マスタは書くアドレスに対して Write 要求を送出し，0x1c から 0x77 までを detect でスキャンしている．そしてスレーブデバイスは Write 要求に応答できる場合，Write 要求に対して SDA を LOW に固定 (ACK) する．マスタは書くアドレス LWrite 要求送出後の SDA が HIGH の場合，NACK(=スレーブデバイスがいない) とし，SDA が LLOW の場合、ACK(=スレーブデバイスがいる) とし，最後にその結果を出力する [14]．このような仕組みで i2cdetect は動いている．つまり，データが送ることができる状態のセンサーのレジスタ値を読み取ることができるのである．

4.3 死活監視

では，i2cdetect を利用して具体的にどのようにしてセンサーの死活監視をするのか．i2cdetect の出力は 4.2 で示したようにデータが送ることができる状態のセンサーのレジスタ値が出力される．つまりこのレジスタ値に対応するセンサーは正常であるということを判別することができるのである．ただしこれはコマンドプロンプトでの標準出力の機能なのでプログラムとしてシステムを構築するためには，標準出力を読み取る必要がある．標準出力から数値へ変換する方法は後述する 5 の中で具体的に説明するものとする．

しかしレジスタ値を取得できただけではどのようなセンサーが繋がっているかがプログラムは把握することができない．そのため本研究では，CSV ファイルを用意し，レジスタ値とセンサーの名前を紐づけできるようにした．紐づけできてないセンサーがある場合，ユーザー側にレジスタ値の入力を求めるようにすることで一度ユーザーに入力させるだけで監視が行えるようにすることが可能になった．その後はプログラムが自動的に CSV ファイルからレジスタ値とセンサーを紐づけし監視を行うようになる．

4.4 異常通知

センサーの死活監視が正常に動作しているだけではユーザーあるいはセンサーを管理しているエンジニアには異常などを伝えることができない．そのため，異常を検知した際にソフトウェアから自動的に通知をするシステムを構築する．本実験では Gmail と Slack を用いて通知を行った．

通知の内容は異常が発生したセンサー，異常が発生した時刻を記述とする．このようにする結果，いつ，どのセンサーからデータが送られていないのかユーザーが認識できるようになる．

IoT ゲートウェイがマイクロサーバであり Linux 系 OS が入っていることで上記の解決策を実施することが可能である．センサーを適切に監視することでセンサーに問題が起きた際に異常が通知され，早急に対応することが可能である．早急な対応をすることは，重大な問題へと発展することを防ぐ，予防の役割もあり，監視をする重要性は大きい．

5. 実装

システム構成図を 2 に示す．

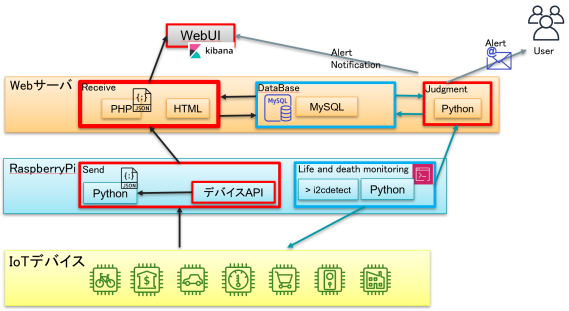

図 2 ソフトウェア構成図

上記にも示したように，対象となるデバイスは全て I^2C 通信プロトコルを使用するものなり，本稿では以下のセンサーを用いて検証を行う．

- BME280
- ADT7410
- SHT31

I^2C 通信プロトコルでは，SDK と SCI のポートしか使わない代わりにセンサーの判別を 16 進数のレジスタ値によって行なっている．どのレジスタに接続されてるかは”i2cdetect”というコマンドを用いて確認することができる．ソースコード 1 に実行結果を示す．

ソースコード **1** i2cdetect 実行結果

```
pi@raspberrypi:~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- 45 -- -- 48 -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- 76 --
```

45,48,76 というのがそれぞれのセンサーのレジスタの値となり，これを元にデータ取得を行う．しかし今回はデータ取得ではなく，センサーの死活監視が目的のためこれらの数字の考慮はしない．レジスタ値とセンサーの組み合わせは以下の通りである．センサーのレジスタ値は製造段階で与えられているものであることを留意して欲しい．

- 45 = SHT31
- 48 = ADT7410
- 76 = BME280

”i2cdetect”コマンドを実行し，これらのレジスタが表示されれば正常にセンサーは RaspberryPi へ接続されていることになる．これらの数値の取得を可能とすれば，ソフトウェア側からセンサーの死活監視が可能になる．実行結果からレジスタの値のみを取得する方法は以下に示す．

(1) 実行結果をスペース区切りで配列に格納する．
(2) レジスタ値以外の要素を取り除く．
(3) String 型で格納されているため，int 型へキャストし数値として扱えるようにする．

以下，配列の操作結果である．

ソースコード **2** レジスタ値のみ取得する

```
実行結果を配列に格納
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
 'a', 'b', 'c', 'd', 'e', 'f', '00:', '--',
'--', '--', '--', '--', '--', '--', '--', '--',
'--', '--', '--', '--', '--', '10:', '--',
'--', '--', '--', '--', '--', '--', '--',
'--', '--', '--', '--', '--', '--', '--',
'--', '20:', '--', '--', '--', '--', '--',
'--', '--', '--', '--', '--', '--', '--',
'--', '--', '--', '--', '30:', '--', '--',
'--', '--', '--', '--', '--', '--', '--',
'--', '--', '--', '--', '--', '--', '--',
'40:', '--', '--', '--', '--', '--', '45',
'--', '--', '48', '--', '--', '--', '--',
'--', '--', '--', '50:', '--', '--', '--',
'--', '--', '--', '--', '--', '--', '--',
'--', '--', '--', '--', '--', '--', '60:',
'--', '--', '--', '--', '--', '--', '--',
'--', '--', '--', '--', '--', '--', '--',
'--', '--', '70:', '--', '--', '--', '--',
'--', '--', '76', '--']

ハイフンを取り除く
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
 'a', 'b', 'c', 'd', 'e', 'f', '00:', '10:',
'20:', '30:', '40:', '45', '48', '50:',
'60:', '70:', '76']

レジスタ値以外の要素を取り除く
[45, 48, 76]
```

このようにしてレジスタの値のみを取得することができる．適切にレジスタの値が取得できればセンサーは正常に動作していることが把握できる．

しかしこれではソフトウェア側からするとただの数値が取れただけでなんのセンサーかということかは認識していない．よって，レジスタ値にセンサーの名前を紐付けしなくてはならない．そのためのプロセスは以下の様にした．

(1) csv ファイルを準備し，レジスタ値の上限である 77 つの”None”(名前を紐付けしていない状態を示している)を”,”（カンマ）区切りで書き込む．以下に状態を示す．
(2) csv ファイルを配列として格納し，配列の要素番号とレジスタ値を同値として扱うことで”None”とレジスタ値が紐付けされることになる．そしてレジスタ値で示された要素が”None” であればユーザーに何のセンサーかを問合せ，入力をさせる．

以上によりレジスタ値とセンサーの紐付けが完了した．因みに csv ファイルの中身は以下の様に更新されている．

(3) レジスタ値とセンサーの紐付けが完了した後，センサーの死活監視に移る．死活監視は予め登録しておいた csv ファイルとその時点での RaspberryPi と I^2C プロトコル通信を行なっているデバイスのレジスタ値

を照らし合わせ，csv ファイルには登録されているセンサーがレジスタ値にない場合，それは何らかの問題があるものとして処理する仕組みになっている．

(4) そして問題が起きた際にはメールや Slack で送信し，ユーザへ認知させる．

以上の流れでセンサーの死活監視を行えるものと本稿では示す．センサー自体に焦点を当てることで，今，何のセンサーがいつ，どこで，正常に動作しているか，又は停止しているが分かるようになることで問題が起きた際の重要な手がかりになる．

6. 評価

6.1 実験環境

本稿では以下の環境で実験を行なった．なおネットワークは研究室内部（ローカルネットワーク）となっている．

- device: RaspberryPi3 modelB+
- OS: raspbian 10.2 (Linux 10)
- cording: Python3.6.1

6.2 死活監視：正常動作時

センサー死活監視における正常動作時の結果を以下に示す．

ソースコード **3** 正常監視

```
pi@raspberrypi:~/tearminal_test $ sudo python3
    input_slack_mail_cmd.py
新しくセンサーを登録しますか？(y/n)
> n
登録は行いません。センサーの監視へ移ります。
SHI31 : OK
ADT7410 : OK
BME280 : OK
問題はありません。
```

6.3 死活監視：異常動作時

センサー死活監視における異常動作時の結果を以下に示す．本実験時は SHI31 と BME280 を回路から外し異常を人工的に発生させた．

ソースコード **4** 異常発生時

```
pi@raspberrypi:~/tearminal_test $ sudo python3
    input_slack_mail_cmd.py
新しくセンサーを登録しますか？(y/n)
> n
登録は行いません。センサーの監視へ移ります。
SHI31 : FALSE
ADT7410 : OK
BME280 : FALSE
SHI31 BME280 に異常があります. メールと
    Slack に通知を送信します.
```

6.2 , 6.3 よりセンサーが正常と異常，それぞれの動作している場合において適切に死活監視ができていることが確認できた．また異常が発生した場合も適切にメールと Slack に通知が送られ，自動通知システムも正常に動作していることが確認できた．1 分間隔でソフトウェアからセンサーを監視しているため，異常が発生してから検知されるまでの時間は最大でも 59 秒であるため，異常検知としても非常に精度が高いシステムではないかと考えられる．またプログラム自体も単純な仕組みで，軽量であるため IoT ゲートウェイとなるマイクロサーバの処理能力を考慮するほど気にしなくて良いという利点もある．これは IoT ゲートウェイのコストを削減することに繋がり，IoT の普及拡大にも繋がると考えられる．

7. 議論

本研究では，I_2C 通信プロトコルを用いたセンサーのみの死活監視であり，I_2C はアクセスコマンドが豊富にあったため監視方法として i2cdetect を用いることは大きな効果があったと考えられる．コマンドを実行させて監視を行うため，Linux 系 OS がインストールされている環境であればどこでも実行することが可能であり，マイクロサーバにインストールされる OS では Linux がトップシェアを誇っているため汎用性も極めて高いと言える．IoT は現在急速に成長を始める，またはもう既に急速な成長を始めその真っ只中にいると考えられており，IoT が医療や軍事などの新しい分野へ参入する前にセンサーへの監視について更に研究を深め，その方法を確立させなければならないのである．また IoT ではハードウェアとソフトウェアが手を結び，互いに密接な関係性をシステムとして持ち合わせることで異常検知や，セキュリティ，そして本研究のテーマであるセンサーの監視をより急速に，そして確実に研究そして実行，展開ができるものであると考える．

8. おわりに

今回は I^2C 通信プロトコルを行うセンサーのみの死活監視であったが，”i2cdetect” というコマンドを使用することで比較的容易にセンサーの動作状態を確認することができた．ただし，センサーが正常か異常かの 2 つの状態のみしか取得できないため，センサーにどのような問題が起きたか，ということは利用者は認知できない．更にソフトウェアからセンサーへ深くデータを得ることができれば，より詳細な動作状態を把握することができると考える．また，本研究では $I2C$ 通信プロトコルでの通信を行うセンサーのみしか死活監視ができていないため，他の通信方式にも適用できるようなセンサー死活監視プログラムを作成することが望ましい．

IoT はモノ (Things) とソフトウェアが繋がるという大きな特徴を持っている．ハードウェアに対して焦点を合わ

せた研究が更に活発になることで IoT に対する信頼が大きく担保されることに繋がるのであろう．

参考文献

[1] Chi, Q., Yan, H., Zhang, C., Pang, Z. and Xu, L. D.: A Reconfigurable Smart Sensor Interface for Industrial WSN in IoT Environment, *IEEE Transactions on Industrial Informatics*, Vol. 10, No. 2, pp. 1417–1425 (online), DOI: 10.1109/TII.2014.2306798 (2014).

[2] Atzori, L., Iera, A. and Morabito, G.: The Internet of Things: A survey, *Computer Networks*, Vol. 54, No. 15, pp. 2787 – 2805 (online), DOI: https://doi.org/10.1016/j.comnet.2010.05.010 (2010).

[3] Vögler, M., Schleicher, J. M., Inzinger, C. and Dustdar, S.: A Scalable Framework for Provisioning Large-Scale IoT Deployments, *ACM Trans. Internet Technol.*, Vol. 16, No. 2, pp. 11:1–11:20 (online), DOI: 10.1145/2850416 (2016).

[4] Zhao, K. and Ge, L.: A Survey on the Internet of Things Security, *2013 Ninth International Conference on Computational Intelligence and Security*, pp. 663–667 (online), DOI: 10.1109/CIS.2013.145 (2013).

[5] Kelly, S. D. T., Suryadevara, N. K. and Mukhopadhyay, S. C.: Towards the Implementation of IoT for Environmental Condition Monitoring in Homes, *IEEE Sensors Journal*, Vol. 13, No. 10, pp. 3846–3853 (online), DOI: 10.1109/JSEN.2013.2263379 (2013).

[6] Porambage, P., Schmitt, C., Kumar, P., Gurtov, A. and Ylianttila, M.: Two-phase authentication protocol for wireless sensor networks in distributed IoT applications, *2014 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 2728–2733 (online), DOI: 10.1109/WCNC.2014.6952860 (2014).

[7] 株式会社ＮＴＴデータ，河村雅人，大塚紘史，小林佑輔，小山武士，宮崎智也，石黒佑樹，小島康平：絵で見てわかる IoT/センサの仕組みと活用，株式会社 NTT (1991).

[8] RaspberryPi: RaspberryPi, RaspberryPi (online), available from ⟨https://www.raspberrypi.org/⟩ (accessed 2020-01-15).

[9] 斎藤秀幸：IoT を検討する前に知っておきたいセンサーの話，CRESCO（オンライン），入手先 ⟨https://www.cresco.co.jp/iot/blog/⟩（参照 2018-05-25）.

[10] ARANGO, M.: Media Gateway Control Protocol (MGCP), *draft-huitema-megaco-mgcp-v1-03.txt*, (online), available from ⟨https://ci.nii.ac.jp/naid/10022085924/en/⟩ (1999).

[11] 雅晴服部，貴仁吉原：サーバ連携による M2M ゲートウェイ設定支援方式の実装とトライアル評価，【C】平成 26 年電気学会電子・情報・システム部門大会講演論文集，pp. 500–504 (2014).

[12] Vandikas, K. and Tsiatsis, V.: Performance Evaluation of an IoT Platform, *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies*, pp. 141–146 (online), DOI: 10.1109/NGMAST.2014.66 (2014).

[13] Molnar, P.: USING I2C SENSORS ON A LINUX VIA A USB AND IIO, petermolnar (online), available from ⟨https://petermolnar.net/linux-i2c-iio-collectd/⟩ (accessed 2018).

[14] 金井 瑛：RasberryPI の I2C コマンド詳解，かないノート（オンライン），入手先 ⟨http://www.hogetan.net/note/memo/⟩（参照 2013）.

