〔論　文〕

搭乗者のバイタル情報を検出するセンシングデバイスの開発

千田 陽介[*1]

Development of Sensing Device for Passenger's Vital Information

Yosuke SENTA[*1]

Abstract

The Intelligent Mobility Laboratory of the Kurume Institute of Technology is investigating and developing smart mobility, a type of wheelchair robot. The mobility is designed to carry the elderly and people with disability; therefore, it must measure the passenger's physical condition (i.e., heart rate and arterial oxygen saturation rate (AOSR)). The total smart mobility system can dispatch paramedics to the mobility that determines if the passenger is ill. Several devices for the physical condition measurement are installed in the mobility; however, regardless of the device, a microcomputer must be used as an embedded system. The AOSR calculation from the sensor value is slightly heavy for the poor calculation power of microcomputers. This study mainly describes the processing of microcomputer programs for obtaining the AOSR with sufficient accuracy for the smart mobility services.

Key Words : Sensor, Smart mobility, Pulse oximeter, Embedded system program

1　はじめに

当学が行っている研究ブランディング事業: 「先進モビリティ技術で多様な人々が能力を発揮できる, Society 5.0 に基づく『いきいき地域づくり』」では, **図 1** のようなスマートモビリティ (車椅子型ロボット) を用いて高齢者や障碍者の社会参加を目指している[(1)]. スマートモビリティにおいて搭乗者の体調を常時監視し, 異常時にはロボット自体が音声で警報を発し付近の者に注意を促したり, 集中管理センタ経由で警備会社等に救護要請を行ったりする機能があってもよい (**図 2**). 体調異常を検出するバイタルサインとして脈拍, 体温, 血圧, 血中酸素飽和度, 心電, 血糖値, 不整脈, 残尿量等様々なものが考えられる. しかし当モビリティはあくまで移動機器であり移動可能な医療計測器ではない. そのため計測が比較的簡単で搭乗者の体調をある程度得られる脈拍と, 時代的に求められている血中酸素飽和度に絞ってこれらを計測する車載システムを検討・試作した.

Fig. 1: スマートモビリティ

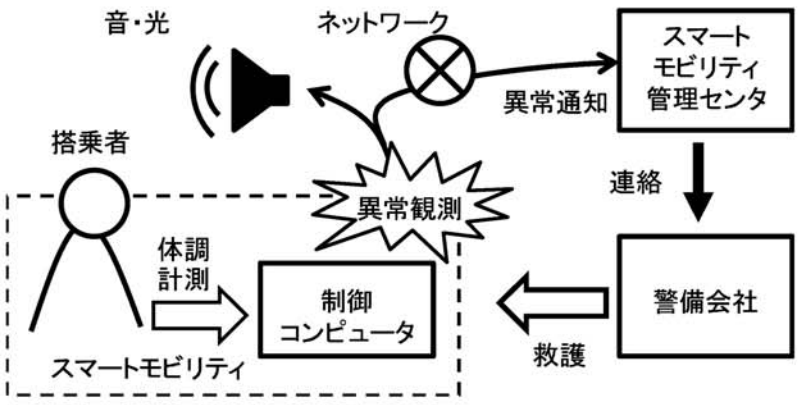

Fig. 2: バイタルセンサの活用事例

[*1] 情報ネットワーク工学科
令和 2 年11月 2 日受理

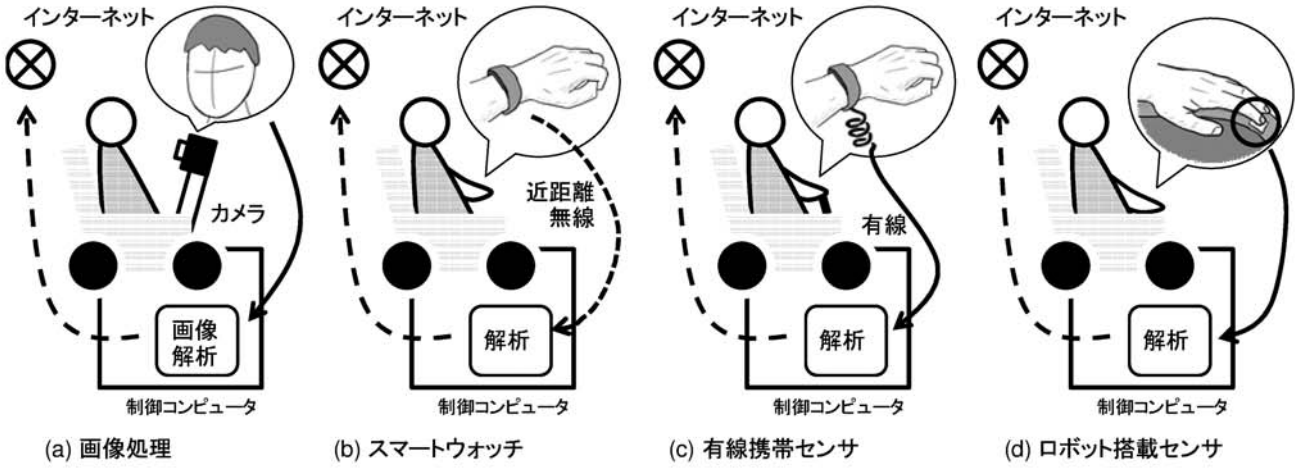

Fig. 3: スマートモビリティ向けバイタルセンサのコンセプト案

2　モビリティにおける体調監視システムの要件

図 2 のようなシステムにおいて, モビリティ上の制御コンピュータは搭乗者との会話や目的地までのルート検索等でネットワーク接続できていることは期待してよい. そのため計測した搭乗者のバイタル情報を管理センタに伝達する部分は, 単に通信プロトコルを合わせるだけであり技術的問題はない. すなわちどのようにして制御コンピュータが搭乗者のバイタル情報 (脈拍や血中酸素飽和度など) を取得するかが鍵となる. 搭乗者のバイタル情報を計測する方法として**図 3** のような様々な方法が考えられる. このうち (a) はカメラを用いて非接触に測定する案である. カメラに映る顔の表面輝度変化から脈拍計測する研究は既になされており [(2)], その成果を適用すればある程度は実現できるであろう. しかしモビリティは屋外を走行することが前提である. 木漏れ日の中を走行するなど外光が激しく変る場面での計測等課題も多くモビリティ用途としては不向きではないかと考えている.

外乱の影響を減らすには接触型センシングが有望である. 図 (b) はそのような案の一つで, 搭乗者にスマートウォッチのようなリストバンド型センサ端末を装着してもらうものである. バンド裏面に適当な接触型センサを取りつけることでバイタル情報が取得できる. 取得した情報を制御コンピュータに伝えるには近距離無線を使えばよい. しかし無線によるワイヤレス化は一見非常にスマートに見えるが, 実運用では, 1. 定期的にセンサ端末を充電しなければならない, 2. 使用者がセンサ端末を装着したままモビリティを降りてしまい紛失することがある, といった問題がある. これはリストバンドをセンサのみの役割に留めず, モビリティのスマートキーや, モビリティを含む生活全体の見守りシステムの中の体調監視, 異常通報ボタンといった役まで担わせれば解決する. しかし逆に言えばそこまで大勢を見極めてシステム全体の中にリストバンド型端末を組み込まなければ実用的なものとならない. 実現へのハードルを落とすにはセンサ端末を有線化すればよい (図 (c)). 電源をモビリィティ上のコンピュータと共用できるのでバッテリの問題はなく, また物理的にモビリィティとユーザがケーブル長以上離れることができないので紛失の心配もない. このように有線化にはワイヤレスには無い利点がある. 有線化の考えをさらに突き進めるとリストバンド型に囚われる必要すら無く, 例えば図 (d) のようにひじ掛け部分にセンサを配置し自然な形で計測する方法もあるであろう.

いずれの方法も一長一短があり, どれが望ましいかは技術的には判断つかない. モビリティを軸としたシステム全体のコンセプト設計から選択すべきと考えている.

3　市販のスマートウォッチを用いた計測

まずはじめに図 3 (b) のような無線リストバンド型のセンサ端末を簡単に実現できないか検討した. このようなセンサ端末は**図 4** のような脈拍計兼血圧計が数千円で売られている. 市販品だけあり外見も非常に完成されており, そのままモビリティシステムの中に取り入れることができれば見栄えも良い. このような市販センサは, 計測値を Bluetooth でスマートフォンへ伝え**図 5** のような専用アプリケーションで表示する. 当システムの最終目的は単にスマートフォンの画面に表示するのでなく, 計測した値に応じて音声を発したり管理センタに通知したりと計測した値を利用することである. そのためには制御コンピュータ上のプログラムに値を取り込む必要がある. 実現が容易で確実な方法は**図 6** のようにスマートフォンの表示画面をカメラで取り込み, 画像解析することであろう. しかし情報処理としては回りくどい. 直接 Bluetooth のデータをプログ

Fig. 4: 市販のセンサ端末

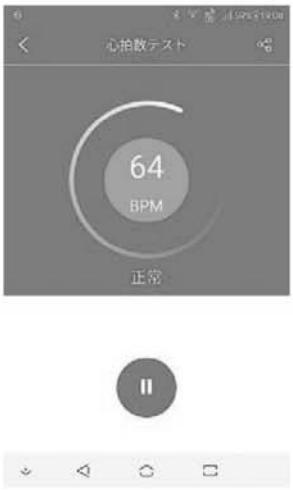

Fig. 5: 専用アプリの表示画面

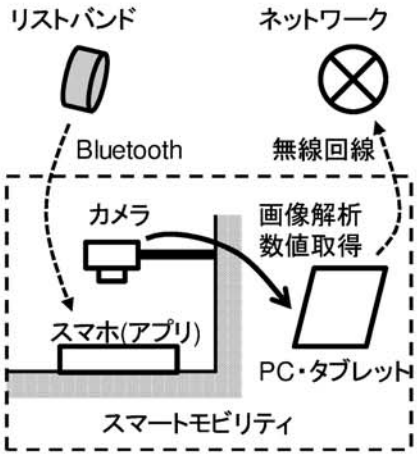

Fig. 6: 画像解析による数値取得案

ラムで受け取ればよいが, プロトコルが不明なためできない. Android の HCI スヌープ機能を用いて, 専用アプリケーションと端末とが通信しているログを採取し解析を試みたが, 予想以上に複雑で良い結果が得られなかった.

Bluetooth デバイスはこの機器のように製品固有のプロトコルを使用するものもあるが, マウスやキーボードといった汎用的機器では共通プロトコル (Generic Attribute Profile: GATT) を用いる. GATT に則った場合あらかじめ準備された適当な API を呼び出すだけで情報を取り出すことができる. GATT の中に Heart Rate Profile という項目もあり, 一部の市販バンド型センサはこのプロトコルに準じている. そのような機器である EPSON PULSENSE PS-100BL 及び iGPSPORT HR60 を用いてみると確かに Visual Basic で記述した PC 上のプログラムから脈拍を受信・表示することができた.

4　脈拍・酸素飽和モニタモジュール MAXREFDES117#

4.1　経皮的動脈血酸素飽和度

経皮的動脈血中酸素飽和度 (以下 SpO_2 値) とは血液の中を流れているヘモグロビンのうち, 何 % が酸素と結合しているかを皮膚を通して計測した値である. 正常値は 96% 以上であり 90% より下がると非常に息苦しく感じる. しかし今年 (2020 年) 世界的な禍となっているコロナウィルス (COVID-19) に罹患すると, 息苦しさを感じないまま SpO_2 値が非常に低い値を示すという事例が紹介されている [3]. これは非侵襲かつ簡易に COVID-19 の罹患可能性を検出できることを意味する. 筆者は医療従事者ではないため, このことの真偽は議論しない. しかし一時期 SpO_2 値を計測する機器 (パルスオキシメータ) が市場で品薄になるなど社会的需要が存在することは事実であり, 市場が求める機能を実現し搭乗者及びその家族に安心をもたらすという意味から当モビリティにも搭載する価値はあると考える.

先述した Bluetooth GATT には Pluse Oximeter Profile という要素もある. この項目に準じた機器を用意できれば容易に SpO_2 値を取得できると考えられるが入手可能な製品の中にそのようなものは無かった. **図 7** は通信機能を持ってない市販

Fig. 7: パルスオキシメータ

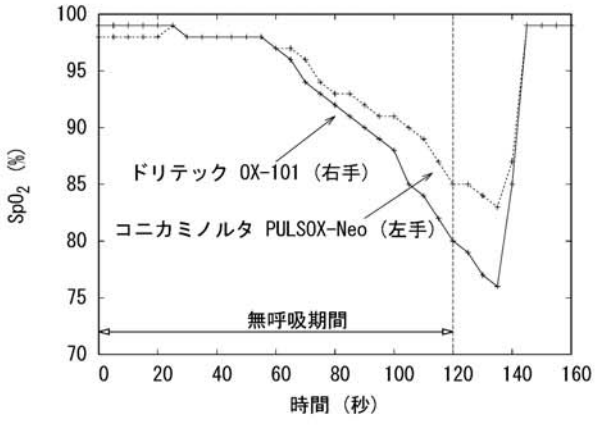

Fig. 8: SpO_2 値の変化

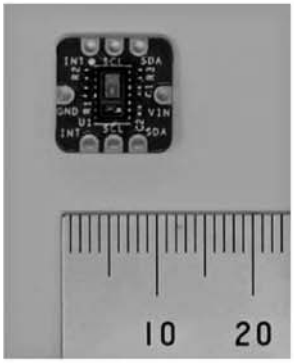

Fig. 9: Maxim MAXREFDES117#

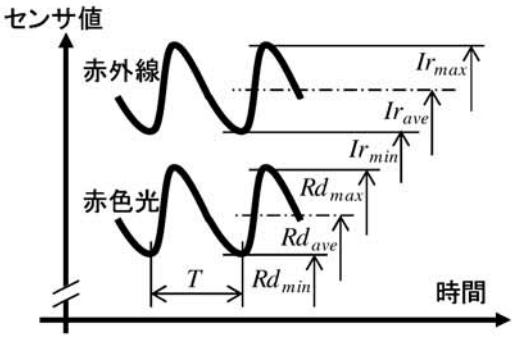

Fig. 10: MAXREFDES117# の出力概要

のパルスオキシメータである (右: ドリテック, OX-101, 左: コニカミノルタ, PULSOX-Neo). 5 章にて詳細を述べるが両者とも厚生労働省より認証を受けている医療機器である. これらは洗濯ばさみのような形状をしており指先に挟んで使用する. 右手人差し指に OX-101, 左手人差し指に PULSOX-Neo を装着して 120 秒間息を止めた時の SpO_2 値の変化を**図 8** に示す. この図より

1. 息を止めて約 60 秒は SpO_2 値がほとんど変化しない
2. 1. の後 SpO_2 値は徐々に下がっていくが機器によって 5 ポイント程の違いがある
3. 呼吸を再開してもおおよそ 15 秒は SpO_2 値は降下し続ける
4. 3. の後, SpO_2 値は急激に復帰する

ことなどが見て取れる. これらは装着する指や被験者といった条件を変えても変わりなかった.

4.2 MAXREFDES117#

図 3 (b)~ (d) のような機器を実現するにあたって, 電子回路を作成できれば形状や機能の面で自由度が出る. そこで市販のセンサ (Maxim MAX30102[4]) を用いてパスルオキシメータを試作した. ここで MAX30102 は 5.6 × 3.3mm の中に 14 個の端子がある小さな素子で, はんだ付けが難しい. また電源も複数の電圧が必要で回路設計が煩雑となる. そこで MAX30102 周りの回路を 12.2 × 12.2mm の基板にパッケージ化した MAXREFDES117# (**図 9**) を活用した. MAX30102 (MAXREFDES117#) は赤外線と赤色光二種類の光を皮膚に当て, それぞれの反射光の強さによって脈拍及び SpO_2 値を計測するものである. 二つの反射光の強度は I^2C 通信を介して**図 10** のような時系列データが得られる. ここから脈拍 (一分間の回数) は

$$HR = \frac{60}{T} \tag{1}$$

という式で得られる (T は一波長の周期). また MdecularD[6] によると, SpO_2 値 (%) は

$$SpO_2 = -45.06z^2 + 30.354z + 94.845 \tag{2}$$

という計算式で得られる. ただし z は

$$z = \frac{\frac{Rd_{max} - Rd_{min}}{Rd_{ave}}}{\frac{Ir_{max} - Ir_{min}}{Ir_{ave}}} \tag{3}$$

という値である. また Ir_{max}, Ir_{min}, Ir_{ave} 及び Rd_{max}, Rd_{min}, Rd_{ave} は一波長での赤外線及び赤色光反射強度の最大値と最小値, 平均値である.

GitHub[7] にはこのセンサに関する Arduino IDE 用のコードが存在し, I^2C 通信によるセンサのアクセス方法や上記計算式の実装例を参照することができる. しかし Arduino は基板サイズが大きく図 3 (b), (c) のような形態に作ることは難しい. そこで Arduino ではなく Microchip 社の PIC マイコン (16F18326) を使うこととした. PIC マイコンを用いることで機能や形状を自由にすることができ, 表面実装品を使えばリストバンド型[8] も実現できる.

PIC16F18326 はシリアル通信モジュール (MSSP: Master Synchronous Serial Port) を 2ch 持っている. MSSP を用いると, プロトコルに則ってソフトウェアで直接ポートの H/L を制御しなくても, 規定のレジスタに規定の値を書き込んだり読み込んだりするだけでシリアル通信が実現できる. シリアル通信には幾つかの種類がある. MAX30102 との通信規格である I^2C もその一つであるし, PC やスマートフォン, タブレットといったコンピュータ組込み機器との通信でよく使われる調歩

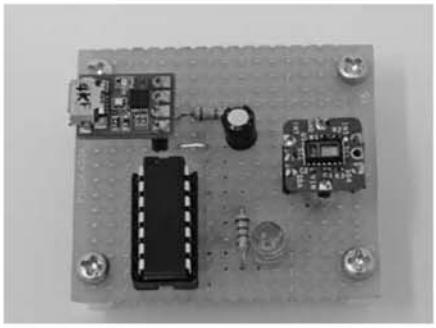

Fig. 11: 試作基板

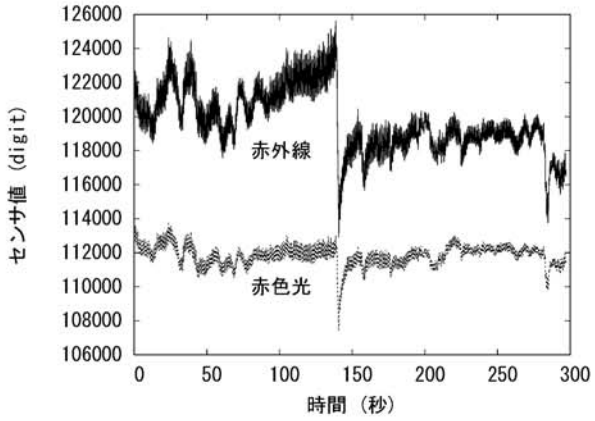

Fig. 12: センサログ

同期方式 (俗に RS232C 方式とも呼ばれる) もその一つである．2ch の MSSP を適切に設定活用することで，簡単に I^2C で受けたセンサ情報を調歩同期方式で制御コンピュータに転送するプログラムを作れる．制御コンピュータと物理的に USB や Bluetooth 等で接続するには市販のシリアル変換モジュールを用いればよい [9, 10]．

図 11 のような回路を作成し，I^2C で読み取ったセンサ値をそのままシリアル通信で PC に転送，保存した結果を**図 12** に示す．この際センサ部分に左手人差し指を乗せて計測した．横軸の単位は秒であり，縦軸は 18bit で得たセンサの出力値である．センサの出力値は式 3 で無次元化されるので物理的にどのような単位かを考える必要はない．この図から

1. 全体を通し約 0.8sec 周期の動きが見受けられる
2. 特に前半部分 15sec 程度の周期の動きがある
3. 時刻 140sec では大きくデータが変化している
4. 赤外線，赤色光の動きはほぼ同じである

という傾向があることが分かる．このうち 1 は脈動によるもの，2, 3 は指をセンサに押しつける力加減が変わったことが原因だと考えている．

120 秒間息を止めた時の計測結果を**図 13** に示す．図 12 と比較しても傾向 1, 2 は同じ現象が起きている．傾向 3 は観測されないが，たまたま体を動かしてしまったことが原因と考えているため問題ない．一方で傾向 4 は前回と異なり赤外線と赤色光で動きが明らかに異なっている．これは息を止めたことによって生じた SpO_2 値の変化が赤外線と赤色光の変化の違いとして現れたのだと考えている．

4.3　SpO_2 値の算出

図 12, 13 に対し各脈拍毎のピーク値を求め，式 2, 3 を適用すれば SpO_2 値を得ることができる．前節の GitHub のサンプルコードには I^2C を用いたセンサ情報の取得だけでなく，SpO_2 値を Arduino 上で算出する処理も含まれていた．そこからセンサデータを SpO_2 値に算出する部分を取り出し図 13 のデータを与えてみた．算出結果と，データを取る際同時に計測し

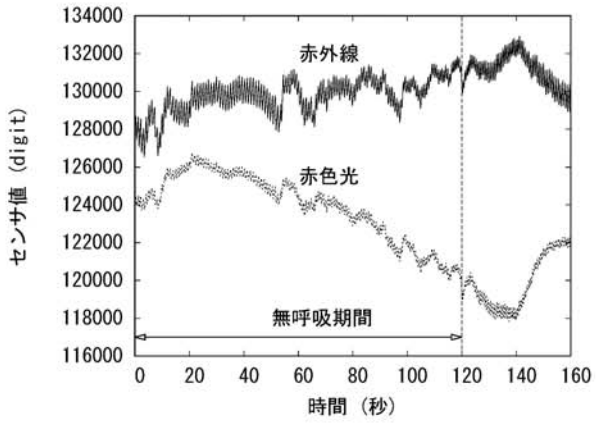

Fig. 13: 無呼吸時のセンサログ

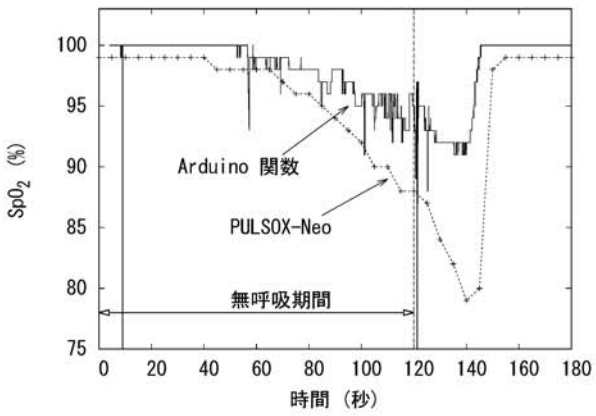

Fig. 14: センサから得た SpO_2 値

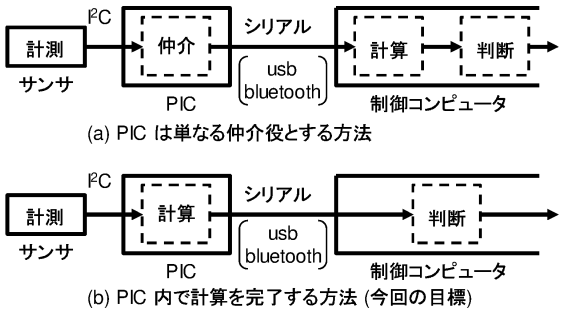

Fig. 15: PIC の役割

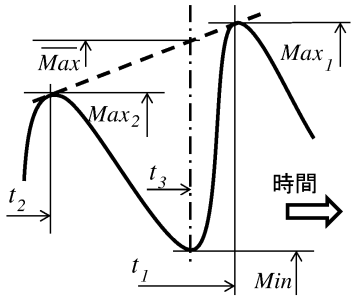

Fig. 16: 線形補間による最大値演算

ていた市販のパルスオキシメータ (PULSOX-Neo: 右手人差し指に装着) が示した値を**図 14** に示す. 図よりサンプルコードが算出した値の変化はやや少なめに出るものの同じような傾向が見える. 市販品に比べノイズが大きいが, これは当コードが一拍毎の SpO_2 値をそのまま出力しているためで, フィルタリングや外れ値除外といった処理を入れれば解決する.

このセンサをスマートモビリティに搭載することを想定すると, 図 2 のようにセンサから管理センタへの経路上に必ず制御コンピュータなど演算能力の高いコンピュータがある. そのため PIC マイコンは仲介役に徹し, I^2C で受けたセンサ情報そのものをシリアル通信で計算力のあるコンピュータに流し, そこで SpO_2 値を計算するという方法もある (**図 15** (a)). しかしもし PIC マイコン単体で SpO_2 値を計算することができれば, 通信機能の無い独立した端末が作れるなど活用の幅が広がる. また些細ながら上位コンピュータの計算負荷や無線帯域の節約にもなる. そこで PIC マイコンで SpO_2 値を計算し, 結果だけ上位コンピュータに通知することにした (同図 (b)).

今回用いる PIC16F18326 は Arduino のマイコン ATmega328 とアーキテクチャは異なるものの計算機としての性能は大差ない. そのため GitHub のサンプルコードそのままは活用できないがのアルゴリズムを移植することはできるであろう. しかしサンプルコードはマイコンで動くよう式 3 を整数型で演算したり式 2 の値をテーブル参照で得たりと工夫は見られるものの, 計算のさせ方やメモリ消費の面から非常に無駄が多いものであった. 例えば図 14 から見て取れるよう, 最終出力は 1% 刻みなのに対し, 常時 32bit の整数型を用いている. また式 3 中の各光の最大値は, 前回と前々回の上ピークの値を Max_1, Max_2, その時の時刻を t_1, t_2, 前回の下ピークが起きた時刻を t_3 とすると,

$$\overline{Max} = Max_2 + \frac{t_3 - t_2}{t_1 - t_2}(Max_1 - Max_2) \tag{4}$$

という計算値を使っていた. これは図 12 で表れていたようなセンサドリフトに対応するため, **図 16** のような線形補間を行っていると想像できるが大げさである. より少ない負荷で同程度の精度を実現できれば, その分マイコンの処理能力に余裕ができ, 将来様々な付加機能を端末に追加することができる.

4.4 計算効率化

PIC マイコンの処理は C 言語 (XC-8) で記述した. ここで PIC マイコンは非力な 8bit マイコンであり, アセンブラレベルでは 8 bit 整数の演算しかできない. また除算はもちろん乗算命令も持ってない. そのため C 言語で int 型 (16bit) の計算や乗除算の処理を記述すると, ソフトウェアで実装されることになる (実数型は言うまでもない). 負荷が低い処理を行うにはできるだけ乗除算は用いずシフト命令で代用することや, できるだけ少ない bit 数で計算すること等の配慮が重要である.

そのような配慮を行い作成したプログラムの処理フローを**図 17** に示す. 図中 (a) は一拍とその間の最大値と最小値を求める処理である. センサ入力は Arduino サンプルコードの I^2C 通信処理をそのまま移植したため 18bit で行う. そのため 3 byte からなる 24bit の整数型 (`Ir`, Rd) で受け取る. それらの最大最小値の更新を行い (図ではまとめて `xx` と表記), 赤外線に関してはピーク検出処理のため移動平均を用いてノイズ除去を行う. ただしここでは処理内容を明確化するため移動平均と表記しているが, ピーク検出を行うことが目的なため最後に N で割る必要はなく, 単に過去 N 回分のセンサ値を加算するだけでよい. これは過去 N 回の `Ir` 値をリングバッファで記憶し, 毎回合計値 `IrA` から N 回前のセンサ値を減じ, 今回得たセンサ値を加えれば合計値を更新することができる. ここで `Ir` はたかだか 18bit のデータであり, `IrA` は 24bit の領域を確保しているので $N = 64$ までならオーバフローの心配はない. 実際 N はそれよりもかなり少ない. 求めた合計値 (移動平均値) `IrA` と一つ前の合計値 `IrB`, 二つ前の合計値 `IrC` とを比べ, `IrB` が一番大きければ上ピークである. 上ピークを検出すると後述する SpO_2 値の算出を行った後, 次のピークに備え今回の最大値 `xxPrevMax` の記憶と, 最大値 `xxMax`, 最小値 `xxMin` のリセットを行う.

SpO_2 値の算出は式 3 を適用する. Arduino のプログラムではドリフト対策のため, 最大値は式 4 を用いていたが負荷の

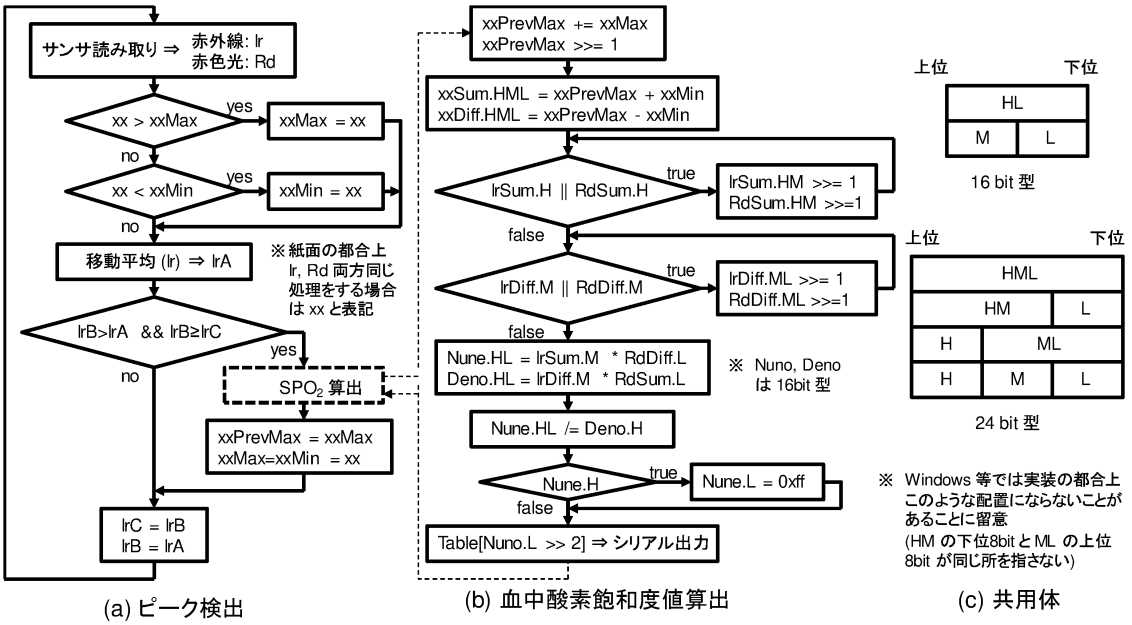

Fig. 17: PIC マイコンの計算フロー

高い積算と除算を要する. 下ピークは上ピークのほぼ中間で発生すると仮定すれば平均: $(Max_1 + Max_2)/2$ で済む. さらに

$$Ir_{ave} = \frac{Ir_{max} + Ir_{min}}{2},\ Rd_{ave} = \frac{Rd_{max} + Rd_{min}}{2} \tag{5}$$

と考え, $Ir_{sum} = Ir_{max} + Ir_{min}$, $Ir_{diff} = Ir_{max} - Ir_{min}$, $Rd_{sum} = Rd_{max} + Rd_{min}$, $Rd_{diff} = Rd_{max} - Rd_{min}$, とおくと, 式 3 は

$$z = \frac{Rd_{diff} \cdot Ir_{sum}}{Rd_{sum} \cdot Ir_{diff}} \tag{6}$$

と変形できる. ここで z から SpO_2 値を得る式 2 は $z = 0.3368$ を頂点とする上に凸な二次曲線であり, $z = 1$ の時 $SpO_2 = 80.139$ である. これより $z > 1$ とは SpO_2 値 80% 以下の, とにかく異常だと断定していい状態である. スマートモビリティにおいて搭乗者が一定レベルを超えた異常事態になった場合, その旨を管理センタに告知するだけでよく具体的な数値を求める必要はないであろう. すなわち $z = 0 \sim 1$ 区間のみ SpO_2 値を求めても実用上問題ない. この区間において式 6 の微分値の絶対値は $z = 1$ の時最大値 14.706 を取る. これは z を 5bit の精度 (1/32) で実装しても, 0.5pt の精度で SpO_2 値が求められることを意味する. これより式 6 の Ir_{sum}, Ir_{diff}, Rd_{sum}, Rd_{diff} は 8bit の整数型で表現しても十分実用的だと言える.

この考えに基づいて記述した計算フローを図 17 (b) に示す. まず前回と今回の最大値の平均と, 式 5 に基づく今回の反射波の平均を求める. 計算結果を前回の最大値 (`xxPreMax`) に格納しているのはメモリ節約のためである. 次に最大値と最小値の和及び差を求める. 元データである `xxMax` や `xxMin` は 24bit 型整数なので和や差も一旦 24bit 整数型に格納しなければならないが, 先述したように精度さえあれば 8bit で十分である. 24bit と 8bit の乗除算の計算負荷は単純計算で 9 倍もの差があるためシフト命令を活用して 8bit 化する. ここで図 12, 13 から和は 16bit で表現できる数 65535 以上, 差は 65535 以下の数になると予想される. すなわち和は上位 16bit の中に, 差は下位 16bit の中に必要な情報が格納されていると考えてよく, 和の下位 8bit や差の上位 8bit までシフト処理するは無駄である. 24bit の整数に対し, 上位もしくは下位 16bit だけ演算するような処理を C 言語で記述するには共用体を活用すればよい (図 17 (c)). ここではこの共用体 (24bit 型) に対し, 変数名に `.HML` を追記すると 24bit 整数型, `.HM` だと上位 16bit, `.ML` だと下位 16bit を 16bit 型整数として, `.H`, `.M`, `.L` だとそれぞれ上位, 中位, 下位を 8bit 型変数として扱うものにして議論を進める.

和の場合, 求めたい 8bit の情報は上位 16bit `.HM` の中にある. そのため赤外線 `IrSum` と赤色光 `RdSum` どちらかの上位 8bit `.H` が非零の間は `.HM` を右シフトする. これにより最終的に `.M` の中に必要情報が格納される. 赤外線と赤色光同時に右シフトしているのは最終的に除算するため先に計算しても結果が変わらないためである. ここで `IrSum.M` と `RdSum.M` のうち元の数が大きかった方は 8bit 目が必ず 1 になるため 128 以上の数であることが保証される. 小さかった方は 127 以下となるため 7bit 以下の精度となる. しかし元の数値に 4 倍以上の開きが無いと仮定すると 6bit の精度, すなわち 32 以上の数になる. 図 12, 13 などのデータから 4 倍以上の開きが無いことは期待してよいであろう. 同様の処理を差に対しても行う. ただし差の場合は, 求める 8bit の情報は下位 16bit `.ML` の中にあるので, `.ML` を右シフトして最終的に `.L` に情報を格納する.

続いて式 6 における分子分母それぞれを得るために乗算を行う．8bit 同士の積の結果は 16bit となるので図 17 (c) で示した 16bit 型の共用体 (.HL が全体, .H, .L がそれぞれ上位下位 8bit) に格納する．この乗算は「8 bit 目が立っている数 (128 ~ 255)」×「8 bit 目が立っていない数 (32~ 127)」,「8 bit 目が立っていない数」×「8 bit 目が立っていない数」「8 bit 目が立っている数」×「8 bit 目が立っている数」の 3 パターンが考えられる．しかし分母の値は $z < 1$ という前提から第 3 のパターンは考えられない．すなわち分母は 4096~ 65535 の範囲にあるはずで, 上位 8bit Deno.H は 16 以上の数であり, 最低でも 5bit の精度を持つ．そのため 16bit の分子 Nune.HL に対し, 分母上位 8bit Deno.H で割ると, 商 (メモリ節約のため Nune.HL に上書きしている) は z を 256 倍した値でありかつ 5bit の精度は引き継がれたものとなる．ここで商の上位 8bit Nune.H が非零の場合は $z \geq 1$ であることを意味する．この節の冒頭に述べたように $z \geq 1$ の場合は総じて異常事態でありその旨を伝えるだけでよいので $z \to 1$ を意味する 8bit で格納できる最大値 255 を Nune.L に格納する．以上の演算により Nune.L には z を 256 倍した整数値が格納さる．これを式 2 に基づいて解けば SpO_2 値が得られるがそれも計算コストが高い．そこで大きさ 64 のテーブルを作成し, Nune.L を右に二回シフトして参照する．このテーブルは 80 ～ 100% の範囲しか格納する必要がないことから, $255(SpO_2 - 80)/20$ という 8bit 整数の値に変換されたものを納めている．

図 18 に図 14 で示した Arduino 用コードが出力した SpO_2 値と, 当アルゴリズムが出力した SpO_2 値の比較を示す．図のように二つのアルゴリズムが出力する結果に違いはほとんどなく, むしろ Arduino 用コードが 1% きざみで出力しているのに対し, 当コードは約 0.4% きざみで出しているため細かく見える．**図 19** は当アルゴリズムの精度を確認したもので, 当アルゴリズムが算出した結果と, 32bit 整数と (Ir_{sum}, Ir_{diff}, Rd_{sum}, Rd_{diff} 算出) 64bit 実数型 (double 型) を用いて式 6 及び 2 を計算して求めた SpO_2 値の差を示したものである．時刻 120 秒あたりで差が大きくなっているのは当アルゴリズムは $z \geq 1$ (80% 以下) は 80% として出力するためである．それを除くと最大でも 0.8pt 程度の差しかない．想定より 0.3pt 程差が大きいがテーブル参照の影響であろう．線形補間等適切な処理をすればこの差は小さくなると思われるが, 現状十分な精度であり無理に処理を加えることはないと考えている．このように計算負荷をかけて実数型で計算しなくても十分な精度が得られることが分かる．

4.5 校正

図 14 から見て取れるように, Arduino のサンプルコード, すなわち当アルゴリズムと市販のパルスオキシメータとの値は傾向は似ているものの出力値にずれがある．これは式 2 のパラメータ −45.06, 30.354, 94.845 は MdecularD がチューニングした結果であり日本人向けではないことが原因だと考える．百~ 千人規模の日本人データを取り, 最小二乗法等を用いて適切なパラメータを求めれば市販品が出す値に近づくと思える．しかしこれは図 3 (b)~ (d) の形態を取るかによって, 外光の影響やセンサの出力傾向等が異なるため, コンセプトが定まってない今行うべきではない．さらに次章に示す薬機法の関係で正確な数値を出すことはできない．校正に関しては原理的には可能なものの実施する意味はないと考えている．

5 薬機法

前章で議論したように MAX30102 センサを用いてパーセントオーダで SpO_2 値を得ることが出来るようになった．また今回は議論を省いたが脈拍も真値と思われる値を得ることができるようになった．しかしこれらの値を直接スマートモビリティ, もしくは管理センタにあるシステム端末に表示することは難しい．日本には俗に薬機法と呼ばれる法律がある (2014 以前は薬事法と呼ばれていた)．正式名は「医薬品, 医療機器等の品質, 有効性及び安全性の確保等に関する法律」であり, 端的

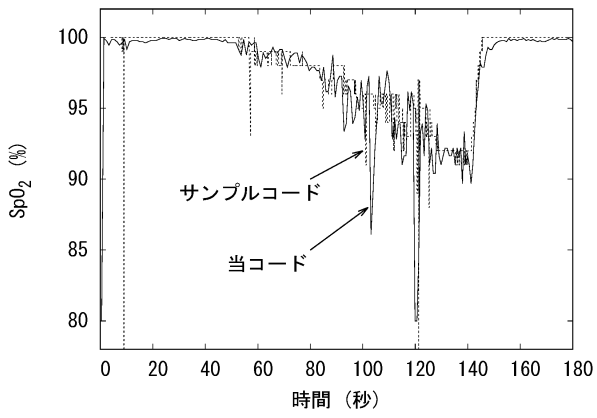

Fig. 18: Arduino 用コードと当アルゴリズム

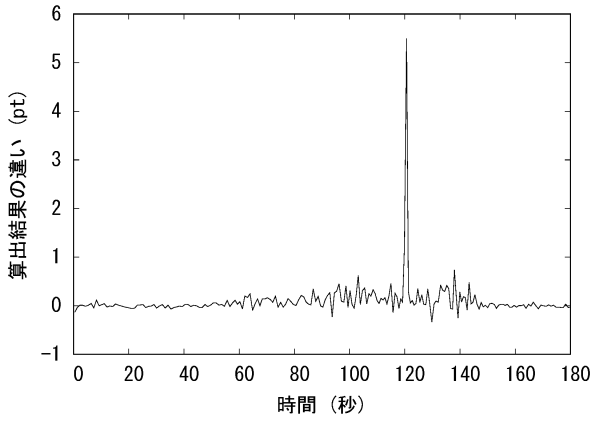

Fig. 19: 実数型との出力比較

には医療品や医療機器を無認可で製造販売してはならないことを定めている. SpO_2 計, 脈拍計は明らかに医療機器であり数値としてユーザもしくはオペレータに提示することは違法行為である. 直接的に数値を提示しなくとも, 単一の計測値を閾値判定して表示や動作を切り替えることもグレーである.

このような問題は例えばスマートフォンやスマートウォッチのアプリケーションなどでも抱えている. これらのアプリケーションでは数値を直接提示するのでなく, 「火星人タイプ」や「水星人タイプ」のようなクラスタリングをしたり○○年齢など非物理的であやふやな単位で表現したりすることで直接医療行為ができないようにしている. 当機器も同様に, SpO_2 値と脈拍数のみならず, さらなるセンサからの情報や会話履歴, 気候などから多角的に搭乗者の体調を推定し, 例えば「いきいき度」のようなあやふやな値として提示する必要があるだろう. 「いきいき度 (仮称)」についてはどのような計算式や `if-then` ルールを用いればよいかは筆者は関与しない. 適当な医療関係者に監修していただいて設計するのが適切であろう.

6　おわりに

当学のスマートモビリティへの搭載を前提としたバイタルセンサについてコンセプトを検討した. また検討された各コンセプトのどのバリエーションにも対応できるよう, 脈拍や血中酸素飽和度が計測できるセンサ回路 (マイコンプログラム含) を試作した. 血中酸素飽和度を得るためには, センサから得られた赤外線と赤色光の反射光強度を少し複雑な計算式に当てはめる必要がある. そこで非力な PIC マイコン単体でも高速に演算できるアルゴリズムを開発した. 算出された値は市販の血中酸素飽和度計 (パルスオキシメータ) と数値は違うもののほぼ同じ傾向が得られた. 校正すればほぼ同じ値が出せると考えているが, 携行コンセプトが定まっておらず, また薬機法の対応が定まっていない段階で行うのは時期尚早であろう.

筆者は民間企業に在籍し組み込み系プログラムの開発をしていたことがある [11]. 今回, その際は半分無意識に行っていたプログラミング時の留意事項を言語化してみた. 半導体技術が発達したためコンピュータの計算能力や記憶量はもはや貴重なものではなくなり, 本稿で述べた数々のテクニックは時代遅れのバッドノウハウの類に見えるかもしれない. 富豪的プログラミング [12] という言葉もある. しかし半導体技術はコンピュータの高性能化という進化の他に, 小型化省電力化という方向へも進化している. 小型化省電力化を極めたコンピュータは相変わらず計算能力やメモリは貴重品である. 本稿で述べた低負荷で必要最小限の精度を保つ演算アルゴリズムに関するテクニックや留意事項は, 今後も変わらず特定分野の IT 産業では必要な知見である. 読者の今後の活動の参考になれば幸いである.

本研究は, 平成 30 年度文部科学省私立大学研究ブランディング事業 (事業名: 先進モビリティ技術で多様な人々が能力を発揮できる, Society 5.0 に基づく「いきいき地域づくり」) の支援を受けており, 謝意を表します.

文　献

(1) 東, “人工知能を搭載した対話型自動運転パートナーモビリティの基本システム開発”, 久留米工業大学研究報告 **43**, (2021)
(2) T. Mori, et. al., “Continuous Real-time Heart Rate Monitoring from Face Images”, The 9th Int. Joint Conf. on Biomedical Engineering Systems and Technologies (2016)
(3) J. Tao, “Early Detection of Silent Hypoxia in Covid-19 Pneumonia Using Smartphone Pulse Oximetry”, J Med Syst. **44**-8, (2020)
(4) Maxim integrated, MAX30102 datasheets (2018), `https://datasheets.maximintegrated.com/en/ds/MAX30102.pdf`
(5) Maxim integrated, Heart-Rate and Pulse-Oximetry Monitor, `https://www.maximintegrated.com/jp/design/reference-design-center/system-board/6300.html`
(6) Instructables circuits, Pulse Oximeter With Much Improved Precision, `https://www.instructables.com/Pulse-Oximeter-With-Much-Improved-Precision/`
(7) GitHub, Arduino C code for MAX30102 pluse oximetry sensor, `https://github.com/aromring/MAX30102_by_RF`
(8) 千田, 青木, “人計測を目的とした安価な無線式センシングデバイスの開発”, 第 3 回 ADADA Japan 学術大会 (2017)
(9) 秋月電子, FT232X 超小型 USB 変換モジュール, `https://akizukidenshi.com/catalog/g/gM-08461/`
(10) 秋月電子, RN42 Bluetooth 2mm ピッチ変換モジュール, `https://akizukidenshi.com/catalog/g/gM-08690/`
(11) 千田, 伊東, “ロボット技術のセンシングデバイスへの展開”, 日本ロボット学会誌, **35**-2, (2017)
(12) 増井, “富豪的プログラミング”, bit, **29**-1 (1997)