Vol.2018-ARC-230 No.1
Vol.2018-SLDM-183 No.1
Vol.2018-EMB-47 No.1
2018/3/7

Raspberry Pi の I/O 制御サーバとライブラリの開発

林勇佑[†1]　早川栄一[†2]

概要：Raspberry Pi を用いたシステム開発において，ケーブルマネジメントの問題があげられる．
そこで本研究ではこの問題を解決するために，無線通信を介して Raspberry Pi の I/O 制御を行うサーバとライブラリの開発を行う．本システムでは，Raspberry Pi は外部センサデータの取得と配信の機能を持つため，システム開発を Raspberry Pi 上で行う必要が無くなる．これにより Raspberry Pi に接続するケーブルは電源と回路の配線のみとなり，ケーブルマネジメント問題を解消することが可能となる．

Development of Raspberry Pi's I/O control server and library

Yusuke Hayashi[†1]　Eiichi Hayakawa[†2]

1. 研究の背景と目的

Raspberry Pi を用いたシステム開発において，Raspberry Pi には電源ケーブル，マウス，キーボード，ディスプレイ出力，回路の配線といった多くのケーブルが接続されており，ケーブルマネジメントの問題があげられる．

また，著者らの所属する大学の講義には Raspberry Pi と回路を利用する演習がある．この演習では受講者がそれぞれ回路を作成し，センサデータを利用したプログラミングを行う．しかしながら，受講者全員が回路を作成することから，人数分の回路部品が必要となってしまう．また，回路を作成している途中に回路部品の破損などもあり，回路部品が足りなくなる場合もあった．

そこで本研究の目的として，ケーブルマネジメントの問題を解決するために無線通信を介して Raspberry Pi の I/O 制御を行うことのできるサーバを開発する．同時に，このサーバには, 複数の人が一つの回路を利用することができ，この回路からセンサデータの取得を行うことを可能とする機能を持たせる．これにより，受講者全員が回路を作成する必要がなくなり，回路部品のコスト削減が可能となる．また，回路の作成時間が減ることにより，講義の時間の中でプログラムに割く時間を増やすことが可能となる．

また，この I/O 制御サーバの利用を簡潔にするためのライブラリの開発を行う. このライブラリを利用することで，従来のプログラムコードを大きく変更することなく実行することを可能とする．

2. 問題分析

2.1 関連手法と問題点

本研究に関連する手法を四つ紹介する．本研究の目的の一つとして，ケーブルの本数を減らすことを挙げた．これを解決する手法として, Raspberry Pi を SSH や VNC で操作をすることがあげられる．この二つはクライアント PC から Raspberry Pi を，ネットワークを通して操作する手法である．これら手法では，Wi-Fi を利用している場合は Raspberry Pi に接続されているケーブルは電源ケーブルだけとなる．

SSH 接続では，CUI を利用することになるため，講義の対象となる初心者には難しいと考えられる．一方で VNC の場合では GUI を利用することができるが，Raspberry Pi では性能不足であり，画面の表示や操作にラグも存在するため操作性が悪くなる．

三つ目の手法として JetBrains 社の Python IDE である PyCharm[1]を利用して, クライアント PC で作成した Python プログラムを Raspberry Pi に転送し，Raspberry Pi 上で実行するという手法である．

四つ目は，Raspberry Pi Zero を用いて，GPIO Expander[2]を利用する手法である．これはクライアント PC に USB で Raspberry Pi を接続することで，Raspberry Pi の GPIO ピンを利用することができるツールである．

しかし，これらの手法では，各個人が回路を作成する必要があり，その回路は同時に一人で利用することしかできない．そのため，いずれの手法も複数の人が同時に一つの回路を利用してセンサデータを取得するという本研究の目的を満たすことはできない．

2.2 要求分析

本システムを開発するにあたっての要求は次の 2 点である．

(1) Raspberry Pi に接続されているケーブル数を減らすために，無線通信を利用して Raspberry Pi に接続されている回路の制御や，センサデータを利用することができるようにする．

(2) 回路部品のコスト削減のために, Raspberry Pi 上のセンサデータを複数のユーザが同時に利用できるようにする．

†1 拓殖大学大学院工学研究科情報・デザイン工学専攻
Graduate School of Takushoku University Information and Design Engineering Course
†2 拓殖大学工学部情報工学科
Takushoku University Department of Computer Science

Vol.2018-ARC-230 No.1
Vol.2018-SLDM-183 No.1
Vol.2018-EMB-47 No.1
2018/3/7

2.3 想定する用途

本研究の想定する用途として，著者らの所属する大学の演習講義を挙げる．この演習講義では，Raspberry Pi と回路を用いて，センサデータを利用したプログラミングを行っている．この講義で本研究の利用を想定した場合，図 1 のようになる．

スイッチや LED など，自分で制御する回路に関しては，それぞれが作成するものとし，温度などのセンサデータを利用する回路に対しては，受講者全員で一つのセンサデータを利用することを想定している．これにより，回路部品数の削減を可能としている．

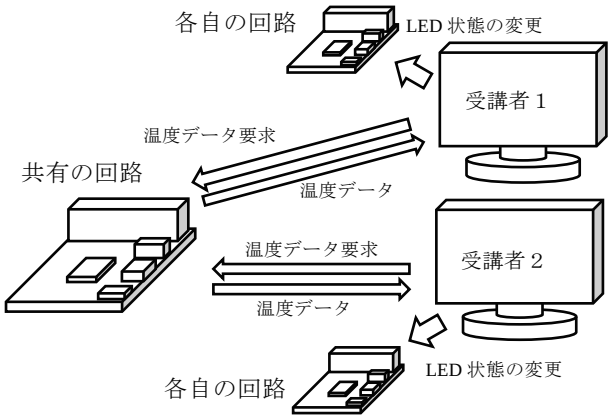

図 1　演習講義での使用例

3. 研究概要

3.1 機能

I/O 制御サーバの主な機能は，次の二つである．

(1) 無線通信を介した外部センサの制御
(2) 複数人が単一センサデータの同時利用

また，ライブラリの主な機能は，次の二つである．

(1) I/O 制御サーバとの相互通信
(2) 複数の I/O 制御サーバとの同時接続

3.2 構成

システム構成図を図 2 に示す．

Raspberry Pi 上で I/O 制御サーバを動作させる．このサーバはクライアントから送信された命令を解析し，センサ状態の変更や，データの取得，クライアントへのデータの送信などを行う．

クライアント向けに, Raspberry Pi 上で動作するサーバに接続し，命令の送信とセンサデータの受信を簡潔に行うためのライブラリを作成する．

Raspberry Pi とクライアント間の通信の手段として本システムでは WebSocket 通信を用いている．

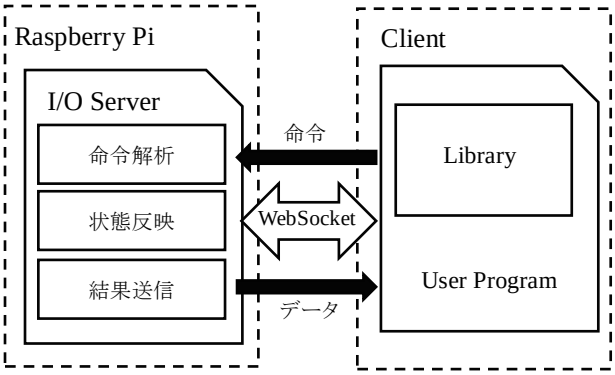

図 2 システム構成図

3.3 I/O 制御サーバ

Raspberry Pi に接続された外部センサを利用し, センサデータの取得と発信を行うための WebSocket サーバを作成する．外部センサは，GPIO で接続されたものを利用する．

このサーバではクライアントから受信したメッセージを解析し，そのメッセージに応じて GPIO の状態を変化や，クライアントへのデータの送信を行う．

また，I/O 制御サーバは WebSocket 通信を行うことで，同時に複数のクライアントとの接続を確立することができる．しかし，複数の人が GPIO の状態を変化させることは混乱を起こすため，GPIO の状態を変化させるような命令は最初に接続したクライアントから受け付けることとする．GPIO の状態を取得するような命令の場合は，GPIO の状態は変化しないため，同時に接続しているクライアント全てから受け付け，データを返す．

3.4 ライブラリ

I/O 制御サーバへの接続と，データの送受信を行うためのライブラリの開発を行う．

このライブラリは命令を I/O 制御サーバに送信し，サーバ側で処理された結果を受信する機能を持つ．

このライブラリを利用することで，I/O 制御サーバへの接続や，GPIO の状態の変更や取得を行うコードを簡潔に書くことができる．このときユーザが記述するコードは Raspberry Pi 標準の RPi.GPIO ライブラリ[3]と同等に書くことができるようにする．これにより，インターネット上に存在するプログラムコードを大きく変更することなく，利用することができる．

また，本ライブラリを利用することで，複数の I/O 制御サーバへ接続することができるため，共有の回路からセンサデータの取得を行い，このセンサデータを用いて自分作成した回路を動かすといったことも可能とする．

4. 実装

4.1 サーバ

I/O 制御サーバには Raspberry Pi 3 を利用し，OS として Debian Stretch with Raspberry pi Desktop を利用する．

Vol.2018-ARC-230 No.1
Vol.2018-SLDM-183 No.1
Vol.2018-EMB-47 No.1
2018/3/7

この Raspberry Pi 3 上で動作する WebSocket サーバは Python3 で作成されており，websocket ライブラリと websocket-server ライブラリ[4]を利用して作成した．この，WebSocket サーバを動作させるにあたってホストの IP アドレスが必要なため，netifaces ライブラリ[5]を利用することで IP アドレスの取得を行っている．また，GPIO の制御には RPi.GPIO ライブラリを利用する．

このサーバはクライアントから受信したメッセージを解析し，次の 9 つの命令を処理することができる．

(1) Raspberry Pi ピンモードの設定
(2) ピン毎の IN と OUT の設定
(3) GPIO output
(4) GPIO input
(5) PWM　周波数とデューティ比変更
(6) PWM　周波数のみ変更
(7) PWM　デューティ比のみ変更
(8) MCP3008 からのデータ取得
(9) GPIO クリーンアップ

Raspberry Pi には GPIO の各ピンの指定方法として，BOARD と BCM の二通りがある. これを指定するための命令が Raspberry Pi ピンモードの設定である．

Raspberry Pi はアナログデータの取得は行えないため，A/D コンバータを利用する必要がある．そのため，A/D コンバータである MCP3008 専用の処理を用意した. この A/D コンバータと Raspberry Pi 間は SPI 通信を行っている．

また，GPIO のクリーンアップを行うことで，GPIO をクリアすることができる．

このサーバでは，最初に接続したクライアントとの接続が切断された時点で GPIO がクリアされるようになっている．これは，GPIO のクリアをしなかった場合に GPIO の状態が保持されたままとなり，次に利用した際にピンモードの設定などが二重に宣言されてしまい，エラーを起こすためである．

本サーバは複数クライアントの同時接続を可能としているが，全てのクライアントから GPIO の状態を変更できてしまうと，利用者は混乱してしまう．そこで，混乱を回避するための排他制御を実装した．

複数クライアントの同時接続時の排他制御を図 3 と図 4 に示す．共にクライアント 1 が最初にサーバへ接続したものとする．

図 3 では，サーバに対して GPIO の状態を変えるような output 命令を送信した場合である．最初に接続したクライアント 1 では，output 命令に対して，命令が通り GPIO の値が変化する．変化したことを伝えるためにサーバはクライアント 1 に対して値が変更されたこと伝える．一方，二番目以降に接続したクライアント 2，クライアント 3 は output 命令を送信しても，サーバからはエラーが返ってくる. これは, 同時に接続しているクライアントがそれぞれ，自由に GPIO の値を変更可能にすると，利用者が混乱してしまうため，最初にサーバに接続したクライアントを管理者とすることで，値を変更できる人と設定する．

図 4 では，サーバに対して GPIO の状態を取得する命令を送信した場合である．Input 命令は GPIO の状態を変化させることなく，状態を取得するだけなので，サーバは全てのクライアントへセンサデータを返す．

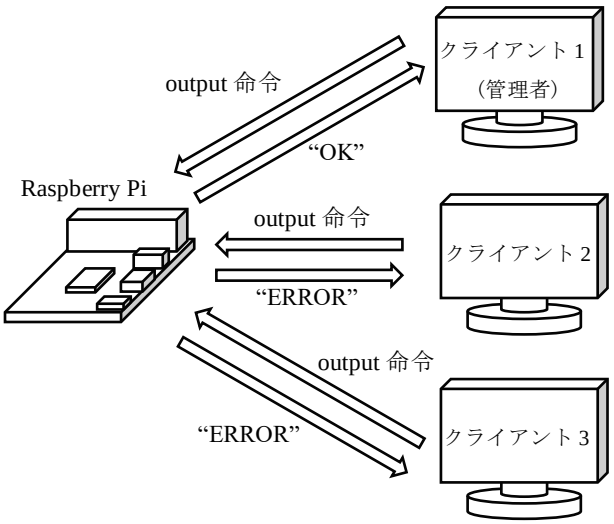

図 3　output 命令の排他制御

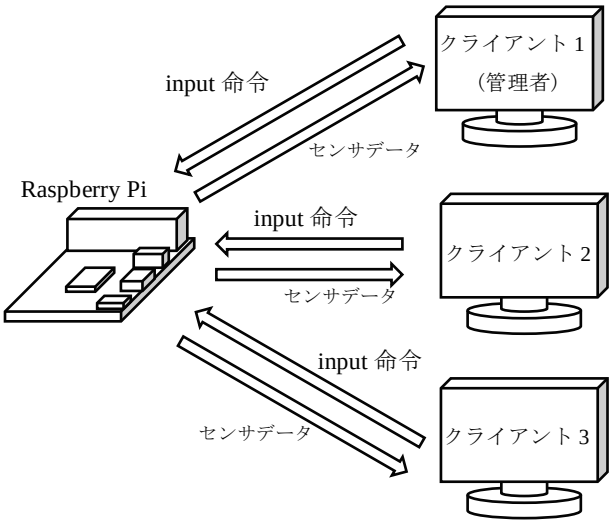

図 4　input 命令の排他制御

9 つの処理のうち，全てのクライアントが実行できる命令は次の二つである．

(1) GPIO input
(2) MCP3008 からのデータ取得

この二つの命令は状態を変化させるものではなく，GPIO の状態の取得や，センサデータの取得のみを行うため，全てのクライアントが使用することができる．

Vol.2018-ARC-230 No.1
Vol.2018-SLDM-183 No.1
Vol.2018-EMB-47 No.1
2018/3/7

4.2 Python ライブラリ

I/O 制御サーバに接続し，メッセージを送信とセンサデータの受信を行うためのライブラリを作成する．このライブラリは Python3 で作成しており，ユーザは Python3 プログラム中でインポートをすることで利用できる．このライブラリでは WebSocket 通信を行うために，websocket-client ライブラリ[6]を利用している．

I/O 制御サーバへの接続や，GPIO の状態の変更や取得を行うコードを簡潔に記述する支援をするライブラリで，このとき記述するコードは Raspberry Pi 標準の RPi.GPIO ライブラリとほぼ同等に書くことができるようにした．これにより，インターネット上に存在するプログラムコードを大きく変更することなく，利用することができるようにしている．

このライブラリによって簡潔に記述することのできる命令は次の 10 通りである．

(1) I/O 制御サーバへの接続
(2) Raspberry Pi ピンモードの設定
(3) ピン毎の IN と OUT の設定
(4) GPIO output
(5) GPIO input
(6) PWM　周波数とデューティ比変更
(7) PWM　周波数のみ変更
(8) PWM　デューティ比のみ変更
(9) MCP3008 からのデータ取得
(10) GPIO クリーンアップ

また，このライブラリを利用することで，複数の I/O 制御サーバに接続することが可能となり，共有の I/O 制御サーバからセンサデータを取得し，そのセンサデータをもとに別の I/O 制御サーバを動作させる事ができる．このようなプログラムの例として，図 5 を示す．

自分の作成した回路には LED が接続されており，温度センサによるデータによって LED を光らせたいとする．従来のように Raspberry Pi と回路だけでは，自分の回路に温度センサを組み込まなければいけない．しかし，本システムを利用することで，温度センサが接続された I/O 制御サーバに接続することで，新たに自分で温度センサ用の回路を作成する必要無く，温度センサのデータを利用することができる．

これにより，演習などにおける回路部品のコストダウンが達成される．また，回路の作成時間を削ることで，プログラミングを行う時間を増やすことができる．

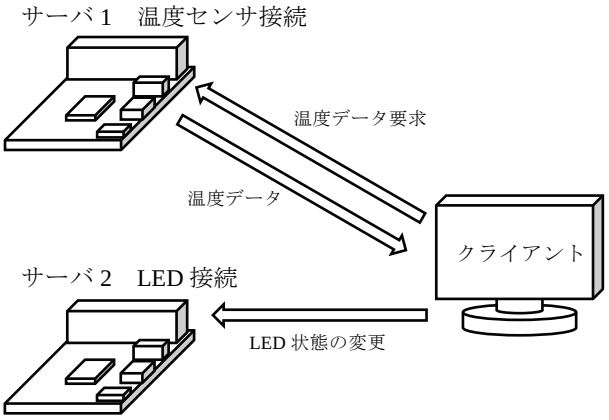

図 5 複数サーバへの接続例

5. 評価

5.1 評価項目

本システムの評価として，次の二点を挙げる．

(1)　プログラムコードの変更点
(2)　応答速度

5.2 プログラムコードの変更点

本システムのプログラムコードの変更点として，Raspberry Pi 3 上で実行させた場合と，本研究のサーバとライブラリを利用した場合で，同じ処理を行うプログラムを作成する．本実験では，Raspberry Pi と回路を利用したプログラミングの入門として利用されることの多い，スイッチを押すごとに LED を点灯, 消灯させるプログラムを作成した．この回路は 21 番ピンに LED を，スイッチを 26 番ピンに接続する．

Raspberry Pi 3 上で実行する従来の手法によるプログラムを図 6 に，本システムを利用した場合のプログラムを図 7 に示す．ここでは，本研究で作成したライブラリの名前を wGPIO としている．

本システムを利用する場合では，WebSocket 通信を行うため, サーバの IP アドレスを指定して接続するコードが追加されている．また，ピンの設定や，output，input 命令も自分で設定したクラス変数が利用されている．

このように GPIO を利用する場合では，プログラムコードに大きな違いはなく，インターネット上に存在する既存のコードを大きく変えることなく利用することができる．

本研究で作成したライブラリによる記述と，従来の RPi.GPIO ライブラリによる記述を比較したものを表 1 に示す．

GPIO 制御では大きな違いは生まれなかったものの，PWM 制御では初めて制御を行うときに，引数を三つ用意するなど, RPi.GPIO ライブラリとは異なる記述を行う必要がある．また，PWM の周波数変更やデューティ比の変更には，従来では変数を用いていたが，本研究のライブラリ

Vol.2018-ARC-230 No.1
Vol.2018-SLDM-183 No.1
Vol.2018-EMB-47 No.1
2018/3/7

ではピン番号を引数で渡しているなど，少し異なる箇所があることがわかる．

```
import RPi.GPIO as GPIO
LED = 21; BTN = 26; old = 0

GPIO.setmode(GPIO.BCM)
GPIO.setup(BTN, GPIO.IN)
GPIO.setup(LED, GPIO.OUT)

while True:
    new = GPIO.input(BTN)
   if new == “0” and old == “1”:
      if GPIO.input(LED) == “1”:
         GPIO.output(LED, 0)
      else:
         GPIO.output(LED, 1)
   old = new
```

図 6　RPi.GPIO ライブラリの場合

```
import wGPIO as GPIO
LED = 21; BTN = 26; old = 0
b = GPIO.GPIO(“ws://{IP address}:9999/”)
b.setmode(GPIO.BCM)
b.setup(BTN, GPIO.IN)
b.setup(LED, GPIO.OUT)

while True:
   new = b.input(BTN)
   if new == “0” and old == “1”:
      if b.input(LED) == “1”:
         b.output(LED, 0)
      else:
         b.output(LED, 1)
   old = new
```

図 7　本研究のライブラリの場合

表 1　実装済み命令の比較

I/O 制御サーバへの接続	
RPi.GPIO では必要がない	Y = wGPIO.GPIO(“ws://{IP address}:9999”)
Raspberry Pi　ピンモードの設定	
GPIO.setmode(MODE)	Y.setmode(MODE)
ピン毎の IN, OUT の設定	
GPIO.setup(PIN, IN:OUT)	Y.setup(PIN, IN:OUT)
GPIO output 命令	
GPIO.output(PIN, STATE)	Y.output(PIN, STATE)
GPIO input 命令	
GPIO.input(PIN)	Y.intput(PIN)
PWM 初期設定	
X = GPIO.PWM(PIN, FREQ) X.start(DUTY)	Y.pwm(PIN, FREQ, DUTY)
PWM 周波数変更	
X.ChangeFrequency(FREQ)	Y.ChangeFrequency(PIN, FREQ)
PWM　デューティ比変更	
X.ChangeDutyCycle(DUTY)	Y.ChangeDutyCycle(PIN, DUTY)
MCP3008 からのデータ取得	
RPi.GPIO では不可	Y.spi(CHANNEL)
GPIO クリーンアップ	
GPIO.cleanup()	Y.cleanup()

5.3 応答速度

本システムの応答速度を計測するための実験を行う．

実験の手法としては，Raspberry Pi 3 上と，本システムの WebSocket 通信を利用した場合それぞれで，GPIO の input 命令を 100 回実行したときの平均の応答速度を計測する．

本システムの実験環境は，サーバとする Raspberry Pi 3 と，クライアント PC は同じ Wi-Fi アクセスポイントに接続し，ローカルネットワークで実験を行った．

実験結果を表 2 に示す．本研究のサーバとライブラリを利用した場合は，1 秒間に約 120 件を処理できるのに対して，Raspberry Pi 3 上で RPi.GPIO を利用した際には 1 秒間に 260 万件を処理することができる．

表 2　応答速度の実験結果

	Raspberry Pi 3 本システム	Raspberry Pi 3 RPi.GPIO Library
平均値	7,966 µs	0.432 µs
最遅値	52,844 µs	5.290 µs
最速値	2,011 µs	0.286 µs
1 秒間の 処理可能件数	約 120 件	約 260 万件

本研究のシステムを利用して実行したときと，Raspberry Pi 3 端末上で実行した場合では応答速度には大きな開きがある．そのため，本システムを利用して，リアルタイム性が求められるプログラムを行うことは困難であると考えられる．しかし，ネットワークを用いたセンサデータの送受

Vol.2018-ARC-230 No.1
Vol.2018-SLDM-183 No.1
Vol.2018-EMB-47 No.1
2018/3/7

信を行っている性質上，時間に厳しいプログラムでの利用は想定していないため問題ないと考える．本システムの用途として挙げた，演習講義等での利用としては，十分な処理速度であると判断する．

6. 終わりに

本研究では，Raspberry Pi の I/O 制御サーバとライブラリの開発を行った. I/O 制御サーバには WebSocket を使用し，無線通信を行うことでケーブルマネジメントの問題を解消した．また，Raspberry Pi をサーバとし，センサデータの取得と発信を行うことで，複数のクライアントが同じセンサデータを同時に利用することを可能とした．すなわち，演習講義では全受講者が回路を作成することなく，共通の回路を利用してセンサデータを利用するプログラムを作成することを可能とした．これにより，回路部品のコストを削減することを可能とした．同時に，回路作成時間を削減することも可能となり，プログラミングに割く時間を増やすことも可能になったと考えられる．

本研究では Raspberry Pi を I/O 制御サーバとした．これにより回路を用いたプログラミングを Raspberry Pi 上で開発する必要はなく, 本研究で作成した Python ライブラリを用いることで，Python が動作する端末ならば，回路を利用したプログラムを開発することができるようになった．そのため，回路を利用しながらも，従来の Raspberry Pi 上で動作させるには処理性能が不足していたプログラムに関しても，クライアント PC の処理性能を利用しながら，回路を利用することができるようになった．

一方で，本研究では I/O 制御サーバを利用するためのライブラリは Python でのみ開発している．そのため，まだ他の言語で利用することができていない．しかし，サーバとクライアント間の通信は WebSocket 通信を利用しているため，Python 以外の言語でも I/O 制御サーバを利用できる．そのため他の言語でもライブラリを作成することで多くの環境や状況で利用できるようになるのではないかと考える．

本研究で作成した I/O 制御サーバでは，GPIO，PWM と A/D コンバータである MCP3008 を利用した SPI 通信をサポートし, センサデータの取得と発信を行えるようにした．しかし，SPI 通信に関しては指定した A/D コンバータ以外で利用することはできないため，用途が限られている．

以上から今後の課題として，WebSocket 以外の通信方法でサーバとクライアントの接続を行うことで，より簡単に接続とデータの取得を行えるようにする必要があると考える．

また, SPI 通信に関しては MCP3008 専用となっているため，より多くのセンサをサポートする必要があるのではないかと考える．

本研究で開発した I/O 制御サーバを利用するためのライブラリは Python 専用となっていることから, 他言語でも同じようにライブラリを開発することで，より利用範囲を拡大できると考える．

参考文献

[1] “PyCharm: Python IDE for Professional Developers by JetBrains”. https://www.jetbrains.com/pycharm/, (参照 2018-01-22).
[2] “GPIO expander: access a Pi's GPIO pins on your PC/Mac - Raspberry Pi”. https://www.raspberrypi.org/blog/gpio-expander/, (参照 2018-01-22).
[3] “raspberry-gpio-python download | SourceForge.net”. https://sourceforge.net/projects/raspberry-gpio-python/, (参照 2018-01-22).
[4] “GitHub - Pithikos/python-websocket-server: A simple fully working websocket-server in Python with no external dependencies”. https://github.com/Pithikos/python-websocket-server, (参照 2018-01-22).
[5] “al45tair / netifaces — Bitbucket”. https://bitbucket.org/al45tair/netifaces, (参照 2018-01-22).
[6] “GitHub - websocket-client/websocket-client: websocket client for python”. https://github.com/websocket-client/websocket-client, (参照 2018-01-22).