レジスタの値からセンサーの種類を識別することによるセットアップ時間の短縮

井出 佑[1]　山本 真也[1]　串田 高幸[1]

概要：IoT 機器はマイクロコントローラ (MCU) とセンサーから構成されている．MCU にセンサーからセンサーデータを取得するためのプログラムを実装，実行することでセンサーデータを取得することができる．このときセンサーの種類に応じて，そのセンサーに対応したプログラムをエンジニアが準備，実装する必要がある．これにより，新規の IoT 機器を導入する際に，センサーデータの取得に時間がかかることが課題になる．この課題に対し，本稿では I2C アドレスと ID からセンサーを識別し，対応するプログラムファイルを MCU に送信するシステムを提案する．この提案では使用するセンサーの名前とセンサーの識別に必要な情報を保存するデータベース (DB) を作成する．そしてセンサーから I2C アドレスと全てのレジスタの値を取得し DB を検索する．検索の結果，一致するセンサーが確認された場合，対応するプログラムファイルを MCU に送信する．評価実験では，提案使用時と不使用時のセンサーデータの取得までの所要時間を測定し，比較することで評価した．実験は東京工科大学の学生 3 人を対象に実験した．1 人目は，提案の不使用時と使用時でセットアップ時間を約 63%削減できた．2 人目は提案の不使用時と使用時でセットアップ時間を約 87%削減できた．3 人目は，提案の不使用時と使用時でセットアップ時間を約 82%削減できた．

1. はじめに

背景

技術の進歩により Internet of Things(以降 IoT とする) が活用される機会が増加している．IoT とは，日常に存在するコンピュータデバイスが埋め込まれたモノ同士が，インターネット経由で相互接続し，データの送受信を可能にするネットワークのことである [1–3]．IoT 機器とは，センサーを組み込んである機器のことを指す．IoT 機器は手動操作を伴わずセンサーデータを収集し，そのデータをインターネット経由で送信することが可能である [1, 4]．IoT の主な活用分野は，スマートシティ，産業用 IoT，コネクテッドビルディング，コネクテッドカー，エネルギーセグメントがある．特に IoT プロジェクト数が増加傾向にあるものは，スマートシティ，コネクテッドヘルス，スマートサプライチェーンセグメントであり，EU とアメリカでは年間 30%以上の増加が見られる [5–8]．また 2025 年までに世界中でネットワークに接続される IoT 機器の数は約 309 億になると予想されており，2030 年には約 1250 億になると予想されている [9]．IoT 機器はセンサーとマイクロコントローラ (以降 MCU とする) からなる．センサーからセンサーデータを取得するためには，センサーを MCU に接続し，MCU にそのセンサーに対応したプログラムを実装，実行する必要がある [10]．

課題

課題は，新規の IoT 機器を導入するたびに，使用するセンサーに対応したプログラムを準備，実装しなければならない点である．例えば，温度センサーである BMP-280 と S-5851A では，温度データが保存されるレジスタアドレスが異なる．さらに，レジスタにアクセスするための I2C アドレスも異なる．これらのことから，同じ温度センサーでも温度を取得する方法が異なるため，各センサーに対応したプログラムを実装する必要がある．図 1 は異なるセンサーがある場合にエンジニアが各センサーに対応したプログラムを準備し，実装している様子である．BMP-280 と S-5851A では温度データを取得する方法が異なるため，エンジニアはセンサーからデータを取る際に，各センサーに対応したプログラムを準備，実装する必要がある．図 1 では MCU として ESP32 を使用しているため，各センサーに対応した Python ファイルを作成し，ESP32 に保存している．

[1] 東京工科大学コンピュータサイエンス学部 〒 192–0982 東京都八王子市片倉町 1404-1

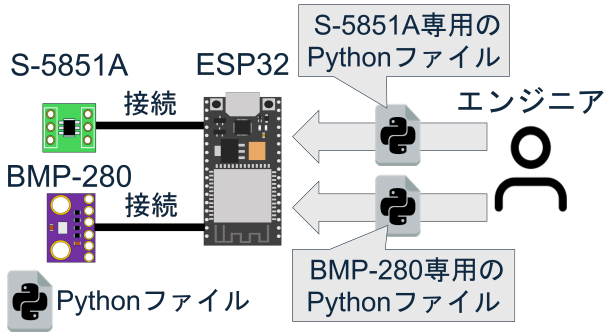

図 1 異なるセンサーがある場合にエンジニアが各センサーに対応したプログラムを準備，実装している様子

各章の概要

第 2 章では本稿の関連研究について説明する．第 3 章では本稿の提案手法について説明する．第 4 章では提案手法の実装について説明する．第 5 章では実験環境と実験の結果の分析について説明する．第 6 章では提案，実験，評価が本稿の課題を解決しているかを議論する．第 7 章では本稿のまとめを行う．

2. 関連研究

ネットワークトラフィックデータに，機械学習アルゴリズムを適用することにより，IoT 機器であるかどうかの識別と，機器の種類の識別を行う研究がある [11]．この研究では 9 種類の IoT 機器と 4 種類の非 IoT 機器を用意して，トラフィックを収集し，教師あり学習を使用しラベル付けすることで，機器の識別を行っている．この研究では，新たな種類の機器が追加された場合，その都度教師データを作成し学習を行わなければならないため，識別可能になるまで時間がかかる．

センサーの識別と故障検出を行うためにフォールカーブを利用したシステムを提案している研究がある [12]．フォールカーブとは，センサーの電源が切れた際に，発生するセンサーの出力データの変化パターンである．これは，センサーデバイス内部の寄生容量に起因する現象である．具体的には，電源が停止して短時間だけセンサーが測定データを出力し続ける際に観測される．フォールカーブはセンサーごとに異なる特徴を持つ．この提案では，センサーごとのフォールカーブの特徴を用いてセンサーの識別と故障検知を行っている．しかし，センサーの種類とメーカーによって，フォールカーブは異なる．そのためセンサーの種類とメーカーごとに解析するプログラムを作成しなければならない．

各 IoT 機器を一意に識別するための識別スキームを判別するために，2 つの学習モデルを組み合わせたハイブリッド DNN モデルを提案している研究がある [13]．DNN とは，脳の神経回路を模したニューラルネットワークを従来の深層学習よりも深い階層に適応させた深層学習の学習方法の 1 つである．1 つのモデルはこの学習方法を使用して，識別子に含まれる文字列を入力として識別スキームを分類している．もう 1 つは CNN を使用して，識別子を 2 次元のポリゴン画像に変換し，識別スキームを分類している．CNN とは DNN の一種で，人の視覚野の機能を模倣した深層学習モデルである．これら 2 つのモデルを組み合わせることで，精度と効率の向上を行っている．この提案では，モデルを 2 つ作成する必要があるため，識別可能になるまで時間がかかる．

3. 提案

本稿では新規の IoT 機器からセンサーデータを取得するまでの時間を短縮することを目的とする．提案方式として，センサーの I2C アドレスと ID からセンサーの種類を判別し，対応するプログラムファイルを返送するシステムを提案する．提案方式の概要として，まず使用するセンサー名，I2C アドレス，ID レジスタのアドレスと値，ID を表すビットの範囲，生データのバイト数を保存するデータベース (以降 DB とする) を作成する．その後，クライアントでセンサーから，I2C アドレスと全てのレジスタの値を取得し，JSON 形式にしてサーバーに送信する．サーバーで受信した JSON 形式のデータを DB で検索して，完全一致するものがあれば，送信するプログラムファイルの一覧を取得し，その一覧の全てのプログラムファイルとセンサー名をクライアントに送信する．クライアントは受信したプログラムファイルを保存する．保存したプログラムファイルを実行し，センサーデータとその生データを取得する．取得した値とセンサー名を JSON 形式にしてサーバーに送信する．サーバーは受信したセンサー名を DB で検索し，生データのバイト数を取得する．受信した生データのバイト数と DB から取得した生データのバイト数を比較して一致する場合，動作確認の終了のメッセージをクライアントに送信する．

提案方式の具体的な内容について，以下のセクションに分けて説明する．

- センサーの I2C アドレスとレジスタの値の取得と送信
- BD の検索とプログラムファイルの送信

提案方式

- **センサーの I2C アドレスとレジスタの値の取得と送信**

図 2 にセンサーからの I2C アドレスと全てのレジスタの値のリストの取得から，取得した値を JSON 形式にしてサーバーに送信するまでの流れを示す．図 2 では使用するセンサーの例として，BMP-280 を使用する場合を示している．

ESP32 が Wi-Fi に接続した後に，ESP32 に接続されているセンサーから I2C アドレスと全てのレジスタの値のリストを取得する．取得したデータを JSON 形式にしてサーバーに送信する．

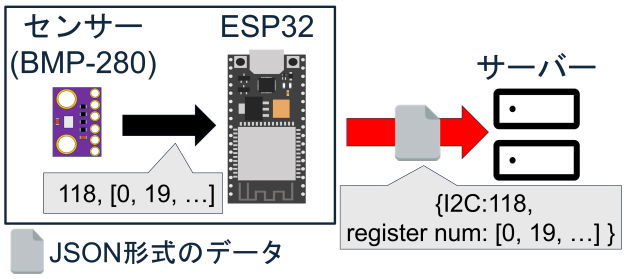

図 2 I2C アドレスと全てのレジスタの値のリストの取得から送信までの流れ

- **BD の検索とプログラムファイルの送信**

ESP32 から送られて来た JSON 形式のデータの内容を DB で検索する．受信したデータと比較して完全一致するものがあれば DB から，送信するプログラムファイルがあるディレクトリのパスを取得し，ESP32 にそのディレクトリにある全てのプログラムファイルを送信する．図 3 は受信した JSON 形式のデータの内容を DB で検索し，対応するプログラムファイルがあるディレクトリを特定し，そこにあるプログラムファイルを ESP32 に送信するまでの流れを示している．図 3 では BMP-280 のライブラリファイルである「Lib_bmp280.py」と，センサーデータを取得するプログラムファイルである「get_data_bmp280.py」を ESP32 に送信している．

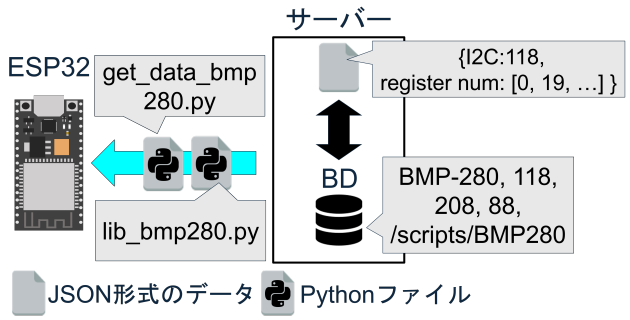

図 3 受信した JSON 形式のデータの内容を DB で検索し，対応する Python ファイルを ESP32 に送信するまでの流れ

ユースケース・シナリオ

東京工科大学コンピュータサイエンス学部の Cloud and Distributed Systems Laboratory(以降 CDSL とする) において，新たに使用するセンサーを ESP32 に取り付ける場合を想定する．提案を適応する前は新たに取り付けるセンサーから，センサーデータを取得するためのプログラムファイルを，エンジニアが準備，実装しなければならない．ここに提案を適用する事でエンジニアがプログラムファイルを準備，実装する工程がなくなり，センサーからセンサーデータを取得するまでの時間が短縮される．図 4 に CDSL におけるユースケースで，本稿の提案を適応したときの ESP32 に，新たに取り付けたセンサーを使用するための Python ファイルを取得するまでの流れを示す．図 4 ではエンジニアが ESP32 に取り付けた，BMP-280 をセットアップしている．ESP32 が Wi-Fi に接続されると，自動で BMP-280 の I2C アドレスとレジスタの値をサーバーに送信し，応答としてセンサーデータを取得するための Python ファイルを受信している．

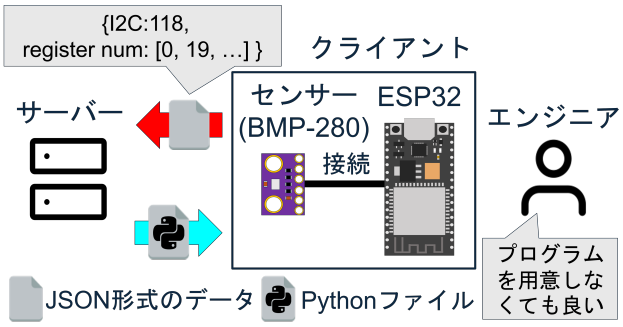

図 4 提案適応後の ESP32 に，新たに取り付けたセンサーを使用するための，Python ファイルを取得するまでの流れ

4. 実装

図 5 にセンサーの識別から必要な Python ファイルの送信までの実装プログラムの処理の流れを示す．

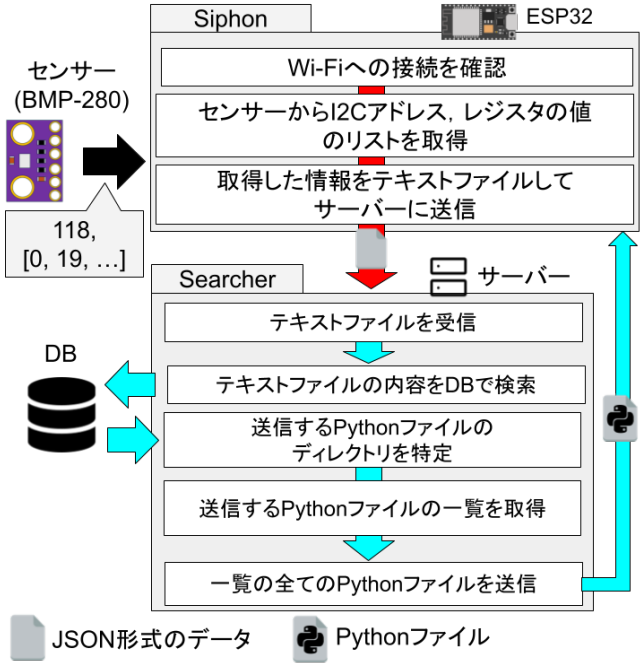

図 5 センサーの識別から必要な Python ファイル送信までの実装ソフトウェアの処理の流れ

ESP32 に接続されているセンサーから，I2C アドレスと全てのレジスタの値のリストを取得し JSON 形式で，サーバーに送信するソフトウェア「Siphon」を実装する．サーバーには，受信した JSON 形式のデータの内容を DB で検

索し，完全一致したときに，送信するプログラムファイルの一覧を取得して，一覧の全ての Python ファイルを ESP32 に返すソフトウェア「Searcher」を実装する．これらのソフトウェアを Python で実装する．DB は MariaDB を使用し，ESP32 とサーバー間の通信には FastAPI を使用する．

実装内容について以下のセクションに分けて説明する．

- DB の内容
- 送信するデータの内容
- DB の検索方法
- プログラムファイルの動作確認

実装内容

- **DB の内容**

表 1 に実装で使用する DB のテーブルの内容を示す．

表 1 DB のテーブルの内容

センサー名 ID レジスタアドレス ID を表すビットの範囲 生データのバイト数	I2C アドレス ID 送信ファイルがあるディレクトリのパス
BMP280 208 [7, 0] [3, 3]	118 88 /scripts/BMP280
BME280 208 [7, 0] [3, 3, 2]	118 96 /scripts/BME280
ADT7410 11 [7, 3] [2]	72 25 /scripts/ADT7410
S5851A 256 [7, 0] [2]	72 256 /scripts/S5851A

カラムは提案で設定した，センサー名，I2C アドレス，ID レジスタアドレス，ID，ID を表すビットの範囲，送信ファイルがあるディレクトリのパス，生データのバイト数を設定する．レジスタから値を取り出す時はバイト単位で取り出す．ID は ID レジスタの 1 バイト目がそのまま ID になる場合もあれば，3 ビット目から 7 ビット目までが ID になる場合もある．このことから，ID を表すビットの範囲を DB に保存しておく必要がある．ID を表すビットの範囲についてはリスト型で表現されており，要素は ID レジスタの値におけるビットの位置を表す．インデックス 0 は最上位ビット (以降 MSB とする) を表し，インデックス 1 は最下位ビット (以降 LSB とする) を表す．表 1 の BMP-280 を例にして説明すると，ID の MSB が ID レジスタの値における 7 ビット目，また LSB が ID レジスタの値における 0 ビット目であることを表している．ID について S-5851A には，ID が無いため ID レジスタアドレスを，レジスタアドレスの最大値に 1 加算した 256 を ID レジスタアドレスとしている．また ID には-1 を設定している．これは，通常入ることがない負の数を入れることで，レジスタから取得した値と区別するためである．NULL にしない理由として，DB の検索の都合上 NULL と int 型では検索方法が異なるため，どのセンサーも全く同じ方法で検索できるように-1 に設定している．

- **送信するデータの内容**

図 6 に BMP-280 の場合に送信する JSON 形式のデータの内容の例を示す．図 6 の中括弧で囲まれた dict 型で表示されいているものが，送信するデータの内容である．キーが”register_num”の要素は全てのレジスタの値のリストである．キーが”I2C_addr”の要素は I2C アドレスである．センサーからは I2C アドレス，レジスタの値を取得することができる．I2C アドレスとレジスタアドレスは 0x00〜0xFF まで存在する．本稿では，センサーを識別するために，I2C アドレスと全てのレジスタの値をセンサーから取得し，JSON 形式でサーバーに送信する．全てのレジスタの値は，インデックスをレジスタアドレス，要素をレジスタの値としてリスト型で取得される．また，ID が無いセンサーの識別のために，リストの最後に 256 を追加し，JSON 形式にする．

```
>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
already Connected. connect skip
0xD0: 0x58
{'register_num': [0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 103, 116,
144, 70, 150, 12, 165, 0, 35, 107, 12
7, 99, 50, 0, 237, 145, 100, 214, 208
, 11, 35, 28, 131, 255, 249, 255, 140
, 60, 248, 198, 112, 23, 0, 0, 14, 0,
0, 0, 0, 0, 0, 0, 0, 0, 51, 0, 0, 192
, 0, 85, 0, 0, 0, 0, 96, 2, 0, 1, 255
, 14, 19, 96, 3, 0, 0, 0, 0, 255, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 88,
1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 128, 0, 0, 128, 0, 0, 0, 0, 0, 256]
, 'I2C addr': 118}
```

図 6 BMP-280 の場合に送信する JSON 形式のデータの内容の例

- **DB の検索方法**

図 7 に DB の検索方法を示す．センサーの識別は DB で I2C アドレス，ID レジスタアドレス，ID，ID の右ビットシフト数が完全一致するものを探す．完全一致にする理由として，BMP-280 と BME-280 は ID 以外全て値が同じ

で，完全一致でないと識別が出来ないためである．検索は，I2C アドレスを受信したものと同じにする．ID レジスタのアドレスと値については，受信した全てのレジスタの値のリストから要素を一つずつ取り出し，その要素のインデックスを ID レジスタアドレス，要素を ID とする．DB のカラムの「ID を表すビットの範囲」のに含まれる全ての値の集合を取得し，集合から要素を一つずつ取り出す．取り出した要素の範囲以外全てが 0 となるマスクを ID にかける．その後，集合から取り出した要素の LSB の分だけ右にビットシフトする．これらのとき集合から取り出した要素を，ID のビットの範囲として検索する．

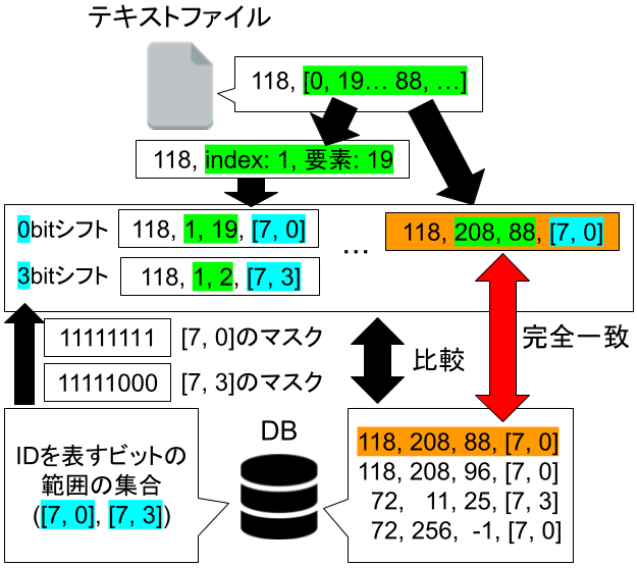

図 7 センサーの識別方法の詳細

- **プログラムファイルの動作確認**

図 8 にプログラムファイルの実行から動作確認の結果の返答までの流れを示す．

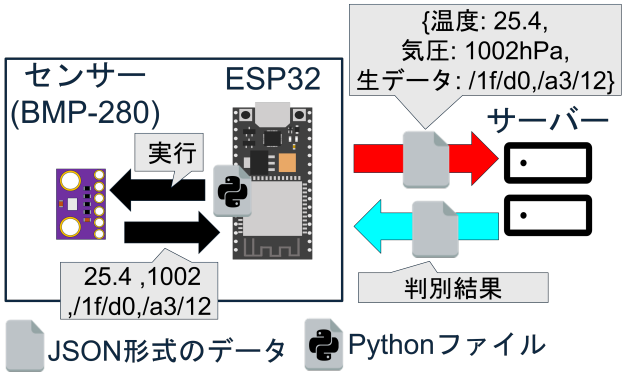

図 8 プログラムファイルの実行から動作確認の結果の返答までの流れ

ESP32 で受信した Python ファイルを実行する．センサーデータとその生データ，センサー名を JSON 形式にしてサーバーに送信する．サーバーで受け取ったセンサー名を DB で検索し，そのセンサーの生データのバイト数を取得する．そして，DB から取得した生データのバイト数と受信した生データのバイト数を比較する．バイト数が一致する場合，通信終了のメッセージを ESP32 に送信する．不一致の場合は，もう一度，センサーデータとその生データ，センサー名をクライアントからサーバーに送信してもらう．

5. 評価実験

評価実験は CDSL の研究室内で行う．サーバーに ESP32 からセンサーデータを送信するまでの時間を，提案を適応する時と適応しない時で比較し評価する．

実験環境

実験環境を図 9 に示す．

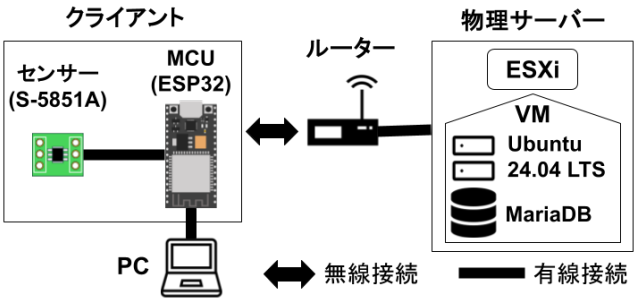

図 9 センサーの識別から必要な Python ファイル送信までの実装ソフトウェアの処理の流れ

クライアント

クライアントは MCU とセンサーで構成されている．MCU には Espressif Systems 社の ESP32 を使用した．センサーは S-5851A を使用した．ESP32 には，MicroPython ファームウェアの ESP32_GENERIC-20230426-v1.20.0 を実装し，MicroPython の開発環境を構築した．また Wi-Fi に接続するための boot.py とセンサーデータをサーバーに送信するための send.py を実装した．提案使用時には，ESP32 に接続しているセンサーの I2C アドレスとレジスタの値を取得する Shiphone.py も実装する．

PC

PC はプログラムファイルの書き込みと，ESP32 への電力供給のために使用する．使用する PC は，Dynabook 社の GCX83/ULE である．また，PC には ESP32 にプログラムを書き込むためのソフトウェア Thonny をインストールしておく．実験では ESP32 を USB 接続し，Thonny を使用して ESP32 内にプログラムファイルを書き込む．

物理サーバー

物理サーバーには米ブロードコム社の VMware 製品である ESXi が運用されている．物理サーバーの ESXi 上に仮想マシン (以降 VM とする) を作成する．VM には Ubuntu

24.04 LTS をインストールする．また，VM にはセンサーを識別するための DB を配置する．DB には MariaDB を使用する．VM には受信した JSON 形式のデータを DB で検索し，必要な Python ファイルを取得して，クライアントに返送する Searcher.py を実装する．

Wi-Fi ルーター

Wi-Fi ルーターは，ASUS 社の「ASUS TUF-AX5400」を使用する．Wi-Fi ルーターと物理サーバーは 1Gbps 対応のケーブルで有線接続されている．

実験結果と分析

評価実験の結果を表 2 に示す．CDSL に所属しているメンバー，筒井 優貴，越後谷 澪，山本 真也の 3 人に対して実験を行った．また，S-5851A から温度データを取得するための，プログラムの作成方法を記した手順書を作成し，プログラムを準備，実装してもらった．その結果，越後谷 澪は提案の不使用時に約 145 秒かかり，使用時は約 53 秒かかった．提案を使用することにより，セットアップ時間を約 63%削減できた．筒井 優貴は提案の不使用時に約 202 秒かかり，使用時は約 25 秒かかった．提案を使用することにより，セットアップ時間を約 87%削減できた．山本 真也は提案の不使用時に約 184 秒かかり，使用時は約 32 秒かかった．提案を使用することにより，セットアップ時間を約 82%削減できた．これら 3 人は全て ESP32 の使用経験がある．提案の不使用時の時間は，約 145 秒〜約 202 秒の約 57 秒の範囲に結果が散らばり，使用時は約 25 秒〜約 53 秒の約 28 秒の範囲に結果がとどまった．このことから，提案の使用時のほうが結果の散らばりが少ないがわかる．これは，提案により個人の能力に関係なく，プログラムの準備と実装が可能になるためであると考える．3 人のうち，越後谷 澪は他二人より，提案使用時の時間が約 2 倍長くなっている．これは，実行するファイルにたどり着くまでに，時間がかかったからであると考える．この問題については，プログラムファイルの動作確認を実装し，実行結果のセンサーデータを自動でサーバーに送信することで，実行すべきファイルを探す手間が省けるため解決できると考える．

表 2 3 人の提案なしとありのときの時間と時間の差

被験者の名前	提案なし	提案あり	時間の差	削減率
越後谷 澪	約 145 秒	約 53 秒	約 92 秒	約 63%
筒井 優貴	約 202 秒	約 25 秒	約 176 秒	約 87%
山本 真也	約 184 秒	約 32 秒	約 152 秒	約 82%

6. 議論

本稿では，DB の検索において，I2C アドレス以外を，レジスタの値のリストのインデックスと要素，DB のカラムの「ID を表すビットの範囲」に含まれる全ての値の組み合わせで，総当たりする方法を取った．この方法では DB のレコード数が増えた場合，検索に時間がかかる可能性がある．このことから，単一のカラムまたは，複数のカラムで区切って検索をし，段階的に対象を絞っていくことで検索を短縮でき，この問題に対応することができる．

現状，新たにセンサーを追加する場合，そのセンサーのデータシートを参照し，I2C アドレス，ID レジスタのアドレスと値，ID の右ビットシフト数，センサーデータの取得方法を調べなければならない．これにより，DB へのセンサーの追加はサーバーにアクセスして手動で行い，センサーデータを取得するためのプログラムファイルは，エンジニアが作成，または探してこなければならない．この問題について，全自動化するのは，不可能である．理由としてセンサーからセンサー名を取得できないためである．このことから，半自動化する案として DB に完全一致するものがなかった場合に，クライアント側のエンジニアにセンサー名を要求する方法を提案する．クライアントからセンサー名を受け取った後は，インターネット経由でそのデータシートを取得する．そして，ID レジスタのアドレスと値，ID の右ビットシフト数，生データのバイト数を生成 AI を使用したデータシートのスキャンにより取得する．また，送信するプログラムファイルは，GitHub からセンサー名で検索して，ヒットしたものを順に取得，動作確認を行い，正常動作したものをサーバーに保存し，そのファイルがあるディレクトリのパスを DB に保存する．この案の懸念点として，生成 AI を使用するため，データシートから取得する値を間違った箇所から取得する可能性がある．また，GitHub からプログラムファイルを取得するため，コードが間違っている可能性がある．例えば，データシートの記載とは異なるレジスタからデータを取得し，センサーデータとして扱う可能性がある．

7. おわりに

本稿の提案では，あらかじめ使用するセンサーの名前，I2C アドレス，ID レジスタアドレス，ID，センサーデータの取得に必要なプログラムファイルが保存されているディレクトリのパスを保存しておく DB を作成する．センサーから I2C アドレスとレジスタの値を取得し DB を検索する．検索の結果，DB に一致するものがあれば，対応するプログラムファイルを全て IoT 機器に送信する．評価実験では，提案使用時と不使用時のセンサーデータの取得までにかかった時間を測定し，比較することで評価する．実験は東京工科大学の学生の 3 人を対象に実験した．1 人目は，提案の不使用時と使用時でセットアップ時間を約 63%削減できた．2 人目は提案の不使用時と使用時でセットアップ時間を約 87%削減できた．3 人目は，提案の不使用時と使用時でセットアップ時間を約 82%削減できた．

謝辞

本稿の執筆にあたり，評価実験に協力していただいた，東京工科大学コンピュータサイエンス学部の越後谷 澪さん，筒井 優貴さんに御礼申し上げます．

参考文献

[1] Mouha, R. A. R. A. et al.: Internet of things (IoT), *Journal of Data Analysis and Information Processing*, Vol. 9, No. 02, p. 77 (2021).

[2] Dorsemaine, B., Gaulier, J.-P., Wary, J.-P., Kheir, N. and Urien, P.: Internet of Things: A Definition Taxonomy, pp. 72–77 (online), DOI: 10.1109/NGMAST.2015.71 (2015).

[3] Madakam, S., Ramaswamy, R. and Tripathi, S.: Internet of Things (IoT): A literature review, *Journal of Computer and Communications*, Vol. 3, No. 5, pp. 164–173 (2015).

[4] Silverio-Fernández, M., Renukappa, S. and Suresh, S.: What is a smart device?-a conceptualisation within the paradigm of the internet of things, *Visualization in Engineering*, Vol. 6, No. 1, pp. 1–10 (2018).

[5] Nižetić, S., Šolić, P., López-de-Ipiña González-de-Artaza, D. and Patrono, L.: Internet of Things (IoT): Opportunities, issues and challenges towards a smart and sustainable future, *Journal of Cleaner Production*, Vol. 274, p. 122877 (online), DOI: https://doi.org/10.1016/j.jclepro.2020.122877 (2020).

[6] Gubbi, J., Buyya, R., Marusic, S. and Palaniswami, M.: Internet of Things (IoT): A vision, architectural elements, and future directions, *Future Generation Computer Systems*, Vol. 29, No. 7, pp. 1645–1660 (online), DOI: https://doi.org/10.1016/j.future.2013.01.010 (2013). Including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond.

[7] Shah, S. H. and Yaqoob, I.: A survey: Internet of Things (IOT) technologies, applications and challenges, pp. 381–385 (online), DOI: 10.1109/SEGE.2016.7589556 (2016).

[8] Wang, J., Lim, M. K., Wang, C. and Tseng, M.-L.: The evolution of the Internet of Things (IoT) over the past 20 years, *Computers Industrial Engineering*, Vol. 155, p. 107174 (online), DOI: https://doi.org/10.1016/j.cie.2021.107174 (2021).

[9] Chowdhury, R. R., Idris, A. C. and Abas, P. E.: Identifying SH-IoT devices from network traffic characteristics using random forest classifier, *Wireless Networks*, Vol. 30, No. 1, pp. 405–419 (2024).

[10] 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).

[11] Meidan, Y., Bohadana, M., Shabtai, A., Guarnizo, J. D., Ochoa, M., Tippenhauer, N. O. and Elovici, Y.: ProfilIoT: a machine learning approach for IoT device identification based on network traffic analysis, *Proceedings of the Symposium on Applied Computing*, SAC '17, New York, NY, USA, Association for Computing Machinery, p. 506–509 (online), DOI: 10.1145/3019612.3019878 (2017).

[12] Chakraborty, T., Nambi, A. U., Chandra, R., Sharma, R., Swaminathan, M. and Kapetanovic, Z.: Sensor Identification and Fault Detection in IoT Systems, *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, SenSys '18, New York, NY, USA, Association for Computing Machinery, p. 375–376 (online), DOI: 10.1145/3274783.3275190 (2018).

[13] Li, X., You, S. and Chen, W.: Automatic Recognition of Identification Schemes for IoT Identifiers based on A Hybrid DNN Model, pp. 1–8 (online), DOI: 10.1109/ICCCN52240.2021.9522273 (2021).

