IoT システムの設計 レポート

園田純太朗

2025 年 12 月 18 日

1,2 日目

基本的に 3 日目以降の内容と重複しており、コードについても教科書とほぼ同じであるため、省略する。ただし、照度の入力についてのみ解説する。照度の取得は以下のコードでできる。

```
int illuminance = analogRead(A0);
```

これは、照度 E と光ダイオードに流れる光電流 I の間には

$$\log_{10} I = \log_{10} E - 5.5 \Leftrightarrow I = 10^{-5.5} E \tag{1}$$

という関係があるが、下図 1 で光電流は $\frac{100\text{k}\Omega}{320\text{k}\Omega} \cdot 1024[V^{-1}] \cdot 1000[\Omega] \cdot E[V] \times$という形で出力されるため、 定数倍の項が打ち消しあって、出力された値は照度とほぼ等しくなっていることがわかる。つまり、出力値に 定数倍を加えて補正する必要はないことがわかる。

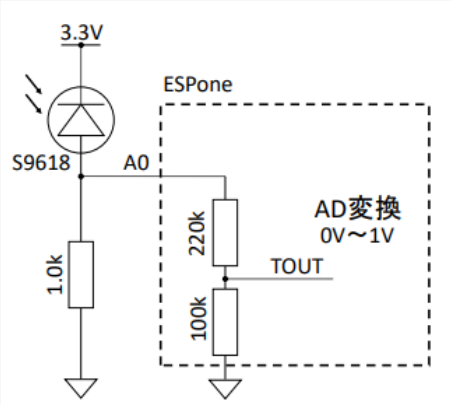

図 1: 照度センサの接続図

3 日目

TCP 接続についても基本的に教科書のとおりである。なお、SSH 接続は linux の ssh コマンドを用いて行った。

次に、これまでのコードを関数群として整理する課題について述べる。 個別の関数についてはこれまでのコードを引数などを変えるだけである。 課題では一つのファイルに記述することが想定されているように思われるが、ファイルが肥大化するため、今後の課題を見据えてファイルの分割を行った。arduino では ino 拡張子を使用し、arduino 言語を用いるが、この arduino 言語は C/C++言語をベースにライブラリを追加したような言語であるため、ファイルの分割は C++とヘッダーファイルを用いて行う。実際のコードの detectPushSWON 関数にかかわる部分の抜粋を以下に示す。ヘッダーファイルでは最初に#ifndefや#define、#endifを用いてインクルードガード処理を施しており、int prev_stat というグローバル変数を extern 修飾子を用いて宣言している。

```cpp
#include "function_list.h"

int prev_stat = HIGH;
bool detectPushSWON() {
  int stat = digitalRead(2);
  if (stat == LOW && prev_stat == HIGH) {
    prev_stat = stat;
    return true;
  } else {
    prev_stat = stat;
    return false;
  }
}
```

リスト 1: function_list.cpp

```cpp
#ifndef _FUNCTION_LIST_H
#define _FUNCTION_LIST_H

extern int prev_stat;
bool detectPushSWON();

#include <Arduino.h>
#endif
```

リスト 2: function_list.h

```cpp
#include "function_list.h"

void setup() {
  Serial.begin(9600);
  pinMode(2, INPUT);
}

void loop() {
  // detectPushSWONのテストコード
  bool isChangedON = detectPushSWON();
  if(isChangedON){
    Serial.println("Pushed!");
  }else{
    Serial.println("not Pushed!");
  }
  delay(1000);
}
```

リスト 3: main.ino

テストケースについては getNPTTime 関数の引数に IP アドレスを入れたときのみ、正しく動作しなかったが、それ以外の単体テストはクリアした。IP アドレスを入力にいれた際に正しく動作しなかったのは入力の際に文字列型で IP アドレスを指定していたからだと思われる。実際、IP アドレスには IPAddress 型という専用の型が用意されているようである。

また、次のように正常に動くテストと、異常系のテストを同時に行うと正常系のテストの方も 0 を返し、正常に動作しなかった。これは正常系のテストが終了し、ポートが解放される前に、異常系のテストが行われてしまい、ポートの初期化に失敗してしまうためであると考えられる。

```
  //WiFiと接続されているときのgetNTPTimeのテストコード
  unsigned long unix_time = getNTPTime("ntp.nict.jp");
  Serial.print("url: ntp.nict.jp :: ");
  Serial.println(unix_time);
  unsigned long unix_time_2 = getNTPTime("www.gutp.jp");
  Serial.print("url: www.gutp.jp :: ");
  Serial.println(unix_time_2);
```

リスト 4: getNTPTime のテストコード

4,5 日目

こちらも基本的に教科書に記述されているコードのうち必要なものを組み合わせたものである。

ただし、3 日目と同様に main 関数にあたる setup(),loop()関数の部分にはできるだけ詳細を入れないようにした。そのため、3 日目に作成した関数群に加えて、以下の 4 つの関数を追加して setup(),loop()関数で使用した。ただし、getCurrentTime 関数については、getCurrentTime 関数内で作成した static char str_time[30]という変数をうまく参照渡しできなかったため、loop 関数内で同様の処理を実装した。

```
//ディスプレイの文字を全消去し、色などの設定をし、カーソルを左上に合わせる。
void initDisplay();
//WiFiに接続する。接続できないときのエラー処理も記述する
bool setupWiFi(const char *ssid, const char *password);
//現在時刻をYYYY-MM-DD HH:MM:SSの形式で返す。
char *getCurrentTime();
//サーバーに接続する。接続できないときのエラー処理も記述する。
bool connectToServer(WiFiClient &client, const char *host, const int port);
//TCP接続し、接続状態を表示する。
void displayCurrentTCPStatus(WiFiClient &client);
```

以下にコードの全容を示す。ただし、エラー処理や print 関数、定数の定義などを省略した疑似コードである。

```
#include "funcList.h"
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

void setup() {
  Serial.begin(115200);

  initDisplay();

  bool isSetupWiFi = setupWiFi(ssid, password);
  if (!isSetupWiFi) {while (1) delay(1000);}

  bool isSyncNTPTime = syncNTPTime(ntp_server);
  if (!isSyncNTPTime) {while (1) delay(1000);}
}

void loop() {
  if (now() / 30 != last_observed_time / 30) {
    int id = getDIPSWStatus();
    char *str_time = getCurrentTime();
    int illuminance = getIlluminance();
    bool isPeopleDetected = getMDStatus();

    WiFiClient client;
    bool isConnected =connectToServer(client, tcp_server_host, tcp_server_port);
```

```
    if (!isConnected) {return;}

    char send_buffer[100] = getCurrentTime();
    client.print(send_buffer);
    displayCurrentTCPStatus(client);

    client.stop();
    last_observed_time = now();
  }

  if (now() / 300 != last_sync_time / 300) {
    syncNTPTime(ntp_server);
    last_sync_time = now();
  }
}
```

また、サーバー側での処理についても簡単に解説する。以下サーバーのコードの抜粋である。

```
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((bind_ip, bind_port))
server.listen(5)

def handle_client(client_socket):
    while True:
        request = client_socket.recv(bufsize)
        try:
            data_str = request.decode("utf-8").strip()
            if data_str.startswith(("GET", "POST" )):#ほかにも列挙している
                client_socket.send(b"ERR\r\n")
                break

            data_fields = data_str.split(',')

            with open("arduino_data.csv", "a", newline='') as f:
                csv.writer(f).writerow(data_fields)

            client_socket.send(b"OK\r\n")

        except UnicodeDecodeError as e:
            client_socket.send(b"ERR\r\n")
        except BrokenPipeError:
            client_socket.send(b"ERR\r\n")
        except Exception as e:
            client_socket.send(b"ERR\r\n")
        finally:
            client_socket.close()

while True:
    client, addr = server.accept()
    client_handler = threading.Thread(target=handle_client, args=(client,))
    client_handler.start()
```

教科書に記述されている部分からの変更点について解説していく。まずは例外処理 try,catch 文を追加することによって、想定外の挙動の時の対応を強化した。そして、ブラウザからのアクセスを拒否するよう 10 行目で HTTP リクエストなどをエラー判定するようにした。そして、9 行目でカンマ区切りの文字列をリストに変化し 14 行目で再度カンマ区切りにした。これらの動作はデータが正常な形式なのかチェックするのに役立っている。こうして入手したデータを with open 分のオプションを a にすることで CSV ファイルに追記することができる。

エラー処理についてはよく起きた二つのエラーについては別で処理した。まず UnicodeDecodeError はそのままの意味でサーバーとクライアントで使用する文字コードが異なっていることで発生した。このエラーの解決のため、9 行目には request.decode(“utf-8”)と UTF-8 を指定した。次に BrokenPipeError はサーバーとクライアントの接続が途切れたときに発生するエラーである。これはサーバーがデータを送信し終わる前にクライエント側で client.stop()を呼び出すことで発生していた。そのため、実際のコードには delay 関数を追加することで解決した。

6 から 10 日目

データの可視化と機械学習を行った。全体像を以下の 図 2 に示す。

まずは 5 日目までの学習成果を利用して ESP One ボードから TCP 接続によって工学部 2 号館にあるサーバーに接続した。このサーバーはリレーサーバーとして利用する。

その後、リレーサーバーからローカル環境のバックエンドサーバーに構築したデータベースへ観測データを HTTP 通信によって送信した。ただし、おそらくリレーサーバーがセキュリティの観点から http による通信を許可していないため、HTTPS 規格での通信を行う必要があった。そのため、一度 ngrok という API を使用した。本郷のサーバーから HTTPS 通信を行ってローカルの ngrok サーバーに接続し、リダイレクトすることでバックエンドサーバーに接続することができた。さらにバックエンドでは Isolation Forest を用いて外れ値を検出し、最新のデータが外れ値かどうかを判定し、その結果もデータベースに登録した。

最後にフロントエンドのコードを作成し、バックエンドのデータベースからデータを取得してグラフに可視化した。

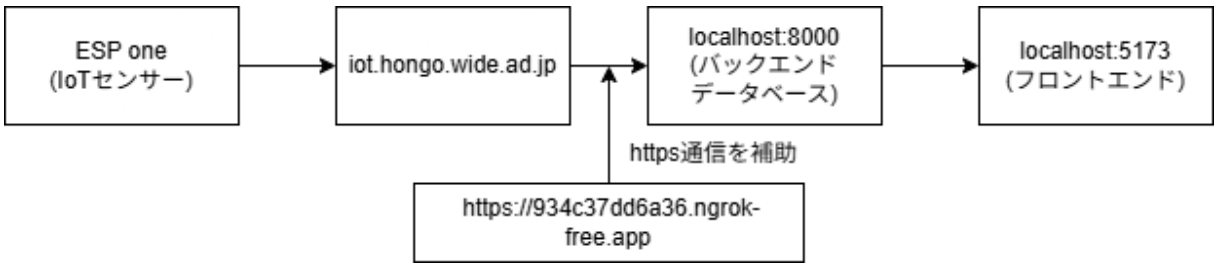

図 2: データの可視化と機械学習の全体像

コード量が多くなってしまったので、バックエンド、フロントエンドの解説については概略のみを示す。 コード全体を参照されたい場合は以下の github を参照していただきたい。また、可視化されたデータのアニメーションについての動画も github の README に記載されているため、そちらを参照していただきたい。 url: https://github.com/junnamuzaurusu/IoTdevice

リレーサーバー

リレーサーバーではサーバー上に CSV ファイルとしてセンサーからのデータを記録することに加え、json 形式のデータをバックエンドサーバーに送信する役割を追加した。

バックエンドサーバー

バックエンドのサーバーは FastAPI を使用して構築した。

参考文献

[1] https://qiita.com/abek21/items/7739163085899b257cb8