0628

【前回の内容について】

研究のアピールポイントを明確にする.

養殖場でできなければ、養殖場と似たような環境を作って実験をして,

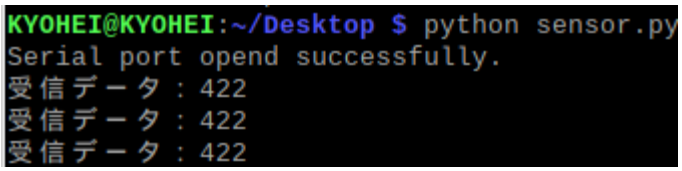
論文では,作った環境と養殖場の比較を事細かく説明し,実際の養殖場と似た環境であることを示さなければならない.　目標は,論文締め切り 9/1 の学会

【センサデータを受信 PC で正しく表示させる】

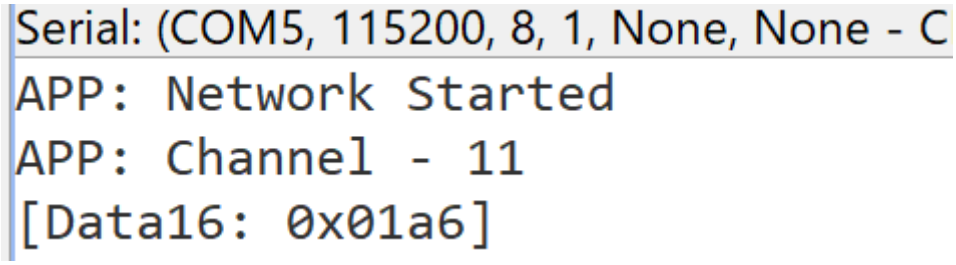
ラズパイ(python)--- Enddevice(NXP) ==wireless==>　Coordinator(NXP)---PC(python)

のフェーズごとにデータ内容を確認した.

・ラズパイ（温度センサー）



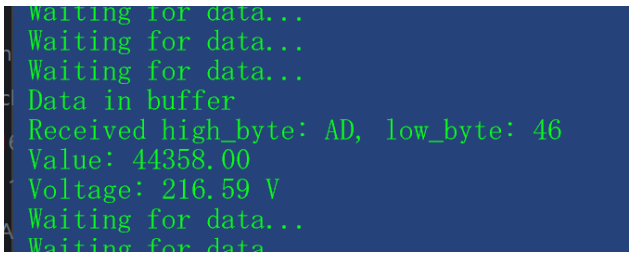
・Coordinator（NXP）



この 16 進数を 10 進数に変換すると,　0x01a6 →　422

ラズパイ(python)--- Enddevice(NXP）は正しい.

・Coordinator が接続されている PC



0xad46 になっていて,データも当然 422 ではない.

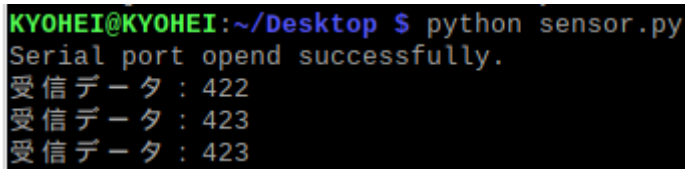
そのため,

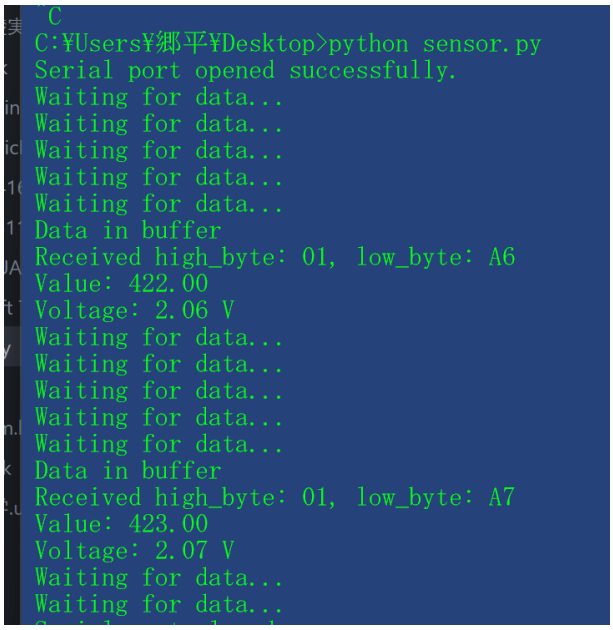
Coordinator(NXP)---PC(python）に問題がある.

Coordinator の NXP において, URAT 通信のコードを以下のように単純にした.

```
u16bytesread = PDUM_u16APduInstanceReadNBO(sStackEvent.uEvent.sApsDataIndEvent.hAPduInst,0,"a\x10",&Rxbyte);
vAHI_UartWriteData(DBG_E_UART_0, Rxbyte[1]); //受信したデータ1つ目をUart通信で送信する関数
vAHI_UartWriteData(DBG_E_UART_0, Rxbyte[2]);//受信したデータ2つ目をUart通信で送信する関数
```

その結果, ラズパイと Coordinator が接続されている PC での表示が以下である.

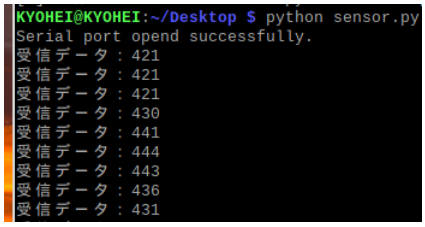




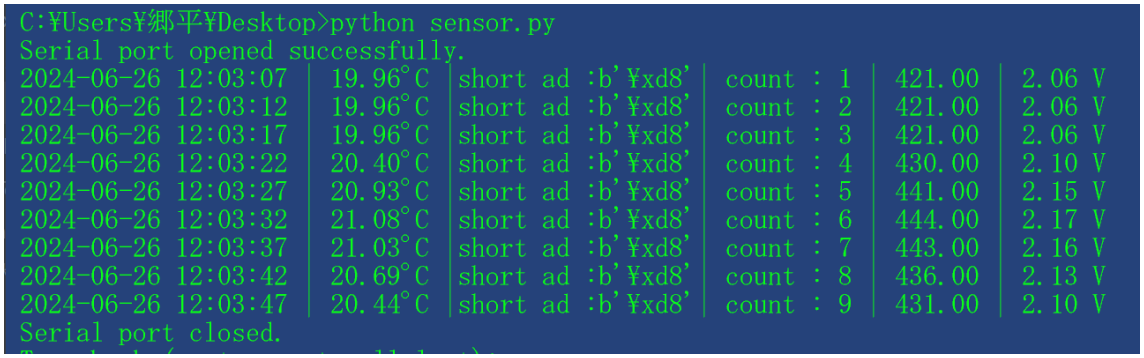
422, 423 と表示され, 成功. また, ボルトに変換したデータも正しい数値である.

以下のように，追加情報を示すようにした．
現在時刻、温度、送信元のアドレス(short)、データの順番

[ラズパイでの結果]



[PC 結果]



[Coordinator の NXP]

```
u16bytesread = PDUM_u16APduInstanceReadNBO(sStackEvent.uEvent.sApsDataIndEvent.hAPduInst,0,"a\x10",&Rxbyte);
i = sStackEvent.uEvent.sApsDataIndEvent.uSrcAddress.u16Addr;
vAHI_UartWriteData(DBG_E_UART_0, Rxbyte[0]); //受信したデータの順番
vAHI_UartWriteData(DBG_E_UART_0, i); //受信したデータの送信元のshortアドレス
vAHI_UartWriteData(DBG_E_UART_0, Rxbyte[1]); //受信したデータ1つ目
vAHI_UartWriteData(DBG_E_UART_0, Rxbyte[2]);//受信したデータ2つ目
```

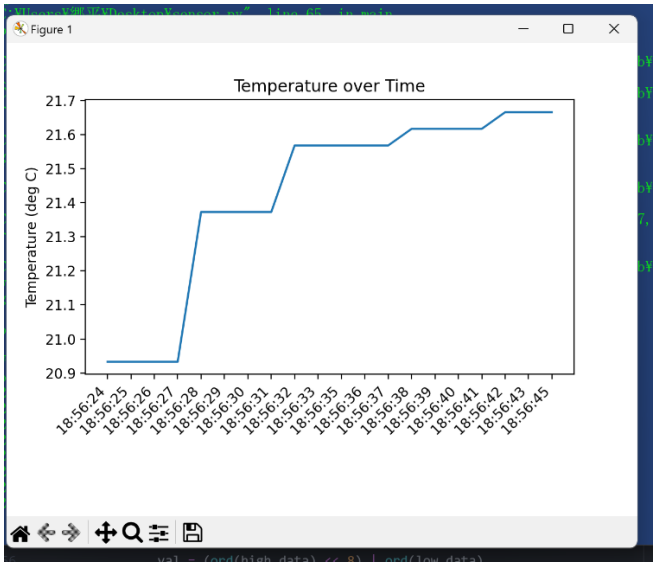
【リアルタイムでの温度センサーのグラフ化】

「pip install matplotlib」で matplotlib をインストールする．以下の URL を参考にし，グラフ化を構築する．

Python と Matplotlib を使用したセンサー データのグラフ化 - SparkFun Learn

その際，温度センサーなどのデータ受信とグラフ更新の並列処理を実装することでリアルタイムのグラフを作成した．

```
C:¥Users¥郷平¥Desktop>python thread.py
Serial port opened successfully.
2024-06-27 18:56:07 | 20.20°C | short ad: b'¥xb0' | count: 1 | 426.00 | 2.08 V
2024-06-27 18:56:12 | 20.20°C | short ad: b'¥xb0' | count: 2 | 426.00 | 2.08 V
2024-06-27 18:56:17 | 20.20°C | short ad: b'¥xb0' | count: 3 | 426.00 | 2.08 V
2024-06-27 18:56:22 | 20.93°C | short ad: b'¥xb0' | count: 4 | 441.00 | 2.15 V
2024-06-27 18:56:27 | 21.37°C | short ad: b'¥xb0' | count: 5 | 450.00 | 2.20 V
2024-06-27 18:56:32 | 21.57°C | short ad: b'¥xb0' | count: 6 | 454.00 | 2.22 V
2024-06-27 18:56:37 | 21.62°C | short ad: b'¥xb0' | count: 7 | 455.00 | 2.22 V
2024-06-27 18:56:42 | 21.67°C | short ad: b'¥xb0' | count: 8 | 456.00 | 2.23 V
2024-06-27 18:56:47 | 21.52°C | short ad: b'¥xb0' | count: 9 | 453.00 | 2.21 V
```



【今後の予定】

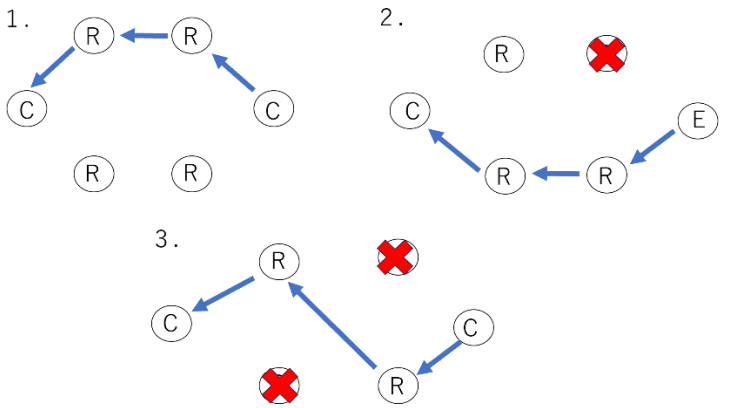
実際に大学内の教室で無線マルチホップネットワークを実装した実験を行う.

→メッシュネットワークを用いて, Ruter の位置や数によるデータ送受信の評価を行う.

→複数台のラズパイが届き次第、Enddevice を増加したネットワークも構築する.

→水槽を用意し，実験を行う.

1．データ送受信やルーティング機能の評価として以下のように実験を行う



2．電波強度を示す LQI を使用していく予定

C:¥NXP¥bstudio_nxp¥sdk¥JN-SW-4163¥Components¥ProductionTestApi¥Include¥jpt.h に，以下の LQI に関する変数と関数があるが，使い方を調査中.

```
106 typedef struct {
107     bool_t bPacketGood;
108     uint16 u16FrameControl;
109     uint16 u16SourceShortAddress;
110     uint16 u16DestinationShortAddress;
111     uint64 u64SourceExtendedAddress;
112     uint64 u64DestinationExtendedAddress;
113     uint16 u16SourcePanID;
114     uint16 u16DestinationPanID;
115     uint8 u8PayloadLength;
116     uint8 u8Payload[127] __attribute__ ((aligned (4)));
117     uint8 u8Energy;
118     uint8 u8SQI;
119     uint8 u8SequenceNumber;
120     uint8 u8LQI;
121     uint8 u8RSSI;
122 } tsJPT_PT_Packet ;
```

```
PUBLIC uint8 u8JPT_GetLQIfromRSSI(uint8 u8RSSI);
```

また，以下の URL は特定のトワイライトのアプリを使用しているが，扱っているパケット内の情報は使えるのではないかと考える．
TWELITE を使ってみた(その 3)｜無線モジュール.com (musen-module.com)

〇パケット内に LQI あり．
〇送信元の論理デバイス，宛先の論理デバイスがあり，ルータ・エンドデバイスの切り替わり，経路も検出可能．
〇タイムスタンプで 1 秒 64 カウントの遅延も測れる．
〇中継フラグで，何回中継したかがわかる．

参考までに PC での python コードを示す．

thread.py

```python
import serial
import time
import datetime
import datetime as dt
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import threading

# グローバル変数として宣言
temperature = None

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
xs = []
ys = []

def animate(i, xs, ys):
    global temperature  # グローバル変数を使用

    if temperature is not None:  # temperatureがNoneでない場合のみ更新
        xs.append(dt.datetime.now().strftime('%H:%M:%S'))
        ys.append(temperature)

        xs = xs[-20:]
        ys = ys[-20:]

        ax.clear()
        ax.plot(xs, ys)

        plt.xticks(rotation=45, ha='right')
        plt.subplots_adjust(bottom=0.30)
        plt.title('Temperature over Time')
        plt.ylabel('Temperature (deg C)')
```

thread.py

```
        plt.title('Temperature over Time')
        plt.ylabel('Temperature (deg C)')

def read_data_from_serial():
    global temperature  # グローバル変数を使用

    try:
        ser = serial.Serial(
            port='COM5',  # デバイスポート番号
            baudrate=115200,  # ボーレート
            parity=serial.PARITY_NONE,
            stopbits=serial.STOPBITS_ONE,
            bytesize=serial.EIGHTBITS,
            timeout=1
        )
        if ser.is_open:  # シリアルポートがオープンされているかチェック
            print("Serial port opened successfully.")

        while True:  # データ受信
            if ser.in_waiting > 1:  # 受信バッファにデータがあるか確認
                count_data = ser.read(1)
                address = ser.read(1)
                high_data = ser.read(1)
                low_data = ser.read(1)
                val = (ord(high_data) << 8) | ord(low_data)
                voltage = val * (5.0 / 1024)
                temperature = (val * 5000 / 1024 - 60) / 100
                current_time = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                print(f"{current_time} | {temperature:.2f}°C | short ad: {address} | count: {ord(count_data)} | {val:.2f} | {voltage:.2f} V")

                time.sleep(5)  # 5秒待機

    except serial.SerialException as e:
        print("Error opening or operating the serial port:", e)
```

```
                voltage = val * (5.0 / 1024)
                temperature = (val * 5000 / 1024 - 60) / 100
                current_time = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                print(f"{current_time} | {temperature:.2f}°C | short ad: {address} | count: {ord(count_data)} | {val:.2f} | {voltage:.2f} V")

                time.sleep(5)  # 5秒待機

    except serial.SerialException as e:
        print("Error opening or operating the serial port:", e)
    finally:
        if ser.is_open:
            ser.close()
            print("Serial port closed.")

def main():
    # 並列処理のためのスレッドを作成
    thread = threading.Thread(target=read_data_from_serial)
    thread.daemon = True  # メインプログラム終了時にスレッドも終了
    thread.start()

    # プロットの設定
    ani = animation.FuncAnimation(fig, animate, fargs=(xs, ys), interval=1000, cache_frame_data=False)
    plt.show()  # プロットを表示

if __name__ == "__main__":
    main()

```