

2020 年度後期 セキュリティ・ネットワーク学実験 2

「センサネットワーク実験」テキスト

担当教員: 山本 寛

1. はじめに

本実験では、IoT (Internet of Things) システムの基盤となるセンサネットワークの構築方法について、基礎的な知識および実装方法を習得することを目的としている。IoT は日本語では「モノのインターネット」と訳されており、これまではインターネットに接続できなかった様々なモノ(例: 家電、自動車など)をインターネットに接続し、情報交換をすることで相互に制御するための技術の総称である。

代表的な IoT システムとして、現実世界を走行する自動車をインターネットに接続し、自動走行やナビゲーションを支援するコネクテッドカーがある(図 1)。このシステムでは、自動車に搭載された様々なセンサーが車の状態を計測(センシング : Sensing)し、計測結果を分析して自動車をどのように制御すれば良いか判断(プロセッシング : Processing)し、その分析結果をもとに自動車を実際に制御(アクチュエーション : Actuation)する。

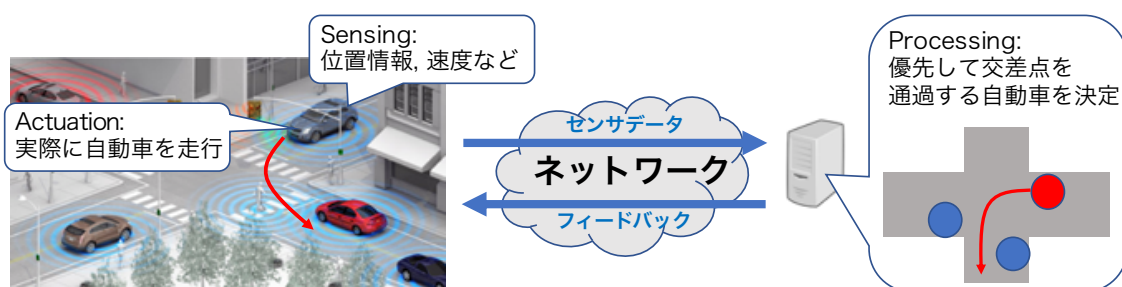


図 1. コネクテッドカーの概要

IoT の基本的な機能は、このセンシング・プロセッシング・アクチュエーションの 3 つであり、他の IoT システムの例である IT 農業の場合には、それぞれ以下ようになる

- センシング : 気温・湿度・土壌水分などを計測
- プロセッシング : 計測した値が作物の生育に適した値か判断し、適していない場合は空調・スプリンクラーの制御が必要と判断
- アクチュエーション : 実際に空調・スプリンクラーを制御

本実験では、センシング・アクチュエーション用の簡単な装置を試作し、コンピュータが持つプロセッシングの機能と連携する IoT システムを開発する。特に、これら 3 つの機能がネットワーク経由で連携する、センサネットワークの構築に取り組む。

2. 注意点

2.1. 本実験の前提知識について

シラバスにも書いているように、本実験は「プログラミング言語」、「プログラミング演習 1」、「データ構造とアルゴリズム」、「プログラミング演習 2」、「コンピュータネットワーク」、「電気電子回路」を受講していることが前提となっているため、これらの科目の内容について実験中に詳しく捕捉することはない。そのため、実験に関する内容の理解に不安のある学生は、これらの科目について必ず復習すること。また、本実験テーマは BYOD による取り組みを推奨する。

2.2. 本実験で作成したプログラムのバックアップについて

自身のコンピュータではなく、大学の実験用コンピュータを利用する場合には、必ず USB メモリを購入して持参すること。実験用コンピュータは、シャットダウンや再起動をすると、それまでに作成したファイルやフォルダが全て削除される。作成したプログラムに関するファイルやフォルダについては、コンピュータをシャットダウン/再起動する前に、必ず USB メモリにバックアップすること。

2.3. 課題とレポートについて

本テキストの 4～7 章で説明している各実験には、【課題〇-〇】と表記された課題が設定されている(7 章は章全体が 1 つの課題)。これらの課題については、1) 理解度を確認するための TA による口頭試問、2) manaba+R へのソースファイル/レポートの提出、といった評価の条件が設定されている。条件を満足していない場合、その課題は評価の対象とはならないことに注意すること。また、各課題に設定されている加点要素を満足している場合には、加点評価を行う。

また、各章の課題の期限は以下の通りである。

- ・ 4 章(【課題 4-1】～【課題 4-3】) - 本テーマ第 3 回実験日の午前 8 時 59 分
- ・ 5 章(【課題 5-1】～【課題 5-4】) - 本テーマ第 5 回実験日の午前 8 時 59 分
- ・ 6 章(【課題 6-1】～【課題 6-2】), 7 章 - 本テーマ第 6 回実験日翌週の午前 8 時 59 分

2.4. ソースコードおよびレポートに関する注意事項

実験に関する、ソースコードやレポート、および、それらの素材として本人が作成すべきものを他の学生に渡す行為(USB メモリやメール、SNS、WEB などを介する共有、印刷媒体の受け渡しなど)は、たとえそれが参考のためであったとしても剽窃幫助にあたるものとみなす。また、受け取る側についても剽窃とみなす。他人のレポートのコピー、または一部改訂したものは、レポートとして受け付けない。これらの行為が判明した場合、試験中の不正行為と同様に扱う。

本実験は、友人と互いに教え合い、学び合うという行為を禁じていない。しかし、これは、プログラムの共同制作を認めているわけではない。プログラムの作成は、あくまで個人に課せられた課題である。プログラムを共同制作した結果、類似したソースコード・レポートが提出された場合についても、剽窃幫助もしくは剽窃と同様に扱う。

レポートの内容が一定の水準に達していないと認められた場合、個別指導を行い、再提出を求める場合がある。この場合の提出期限は、個別に指示する。再提出しなかった場合、レポート未提出として扱うので注意すること。

2.5. サンプルコードなど

実験のサンプルコードおよびレポートのテンプレートは, **manaba+R** から入手すること.

3. IoT システムの基本的な機器構成と本実験の流れ

IoT システムの基本的な機器構成は図 3-1 のようになる。デバイスは現実空間に存在する様々なモノに相当し、センシングとアクチュエーションの機能を提供する。これらのデバイスは直接インターネットに接続することもあるが、基本的には Bluetooth や ZigBee のような近距離無線通信技術によりゲートウェイに接続し、ゲートウェイがデバイスのデータを集約して、有線/無線 LAN や携帯通信(4G/LTE など)によりインターネット上のサーバ/クラウドと通信する。プロセッシングの機能はサーバ/クラウドが主に担当するが、その一部をゲートウェイが代行する構成もある(例: エッジコンピューティング)。

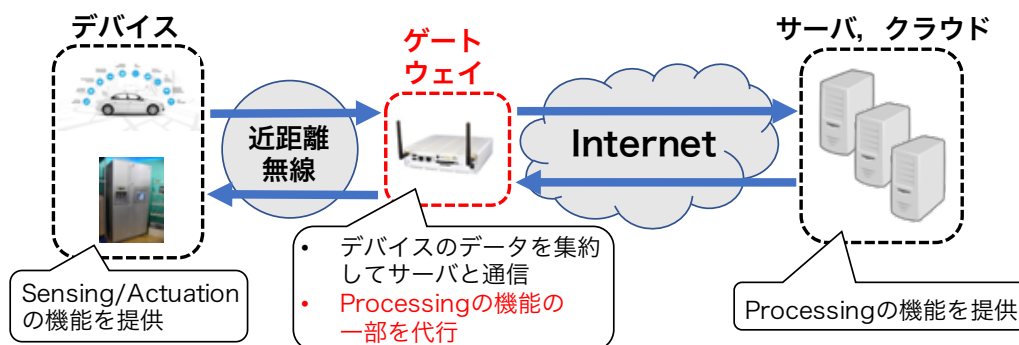


図 3-1. IoT システムの機器構成

本実験ではこのシステム構成を簡略化し、デバイスに相当するハードウェアを代表的な組込みシステム(特定の機能を実現するために家電製品や機械等に組み込まれるコンピュータシステム)である Arduino により構築し、ゲートウェイとサーバ/クラウドに相当するハードウェアを Android スマートフォンにより構築する。また、Arduino と Android スマートフォンは、Bluetooth や ZigBee といった近距離無線によりセンシング/アクチュエーションのためのデータ通信を行う。

初めから Arduino と Android を組み合わせたシステムを構築することは難しいため、まずはハードウェアごとに独立した機能を実装し、その後にハードウェアが連携するシステムを構築する手順で実験を進める。本実験は、基本的には以下のような流れで進める。

(1) Arduino : センシング/アクチュエーションの機能を備えたデバイスの開発

Arduino にセンシングのための装置であるセンサ(照度センサ、音センサなど)と、アクチュエーションの対象となる装置であるアクチュエータ(LED、モータなど)を搭載し、センサによるセンシングの結果をもとにアクチュエーションの種類を選択してアクチュエータを制御(例: 照度センサがある閾値を下回ると LED を ON する)する簡単なデバイスを開発する。

(2) Android スマートフォン : センシング結果を可視化するアプリケーションの開発

プロセッシング機能の基礎である、センシングの結果を可視化するアプリケーションを開発して動作を確認する。開発するアプリケーションは、Android スマートフォンに搭載されているセンサ(加速度、ジャイロなど)が計測した結果を折れ線グラフで表示する。

(3) Android スマートフォン/Arduino : 近距離無線による通信機能の開発

Android スマートフォンと Arduino の間で、近距離無線通信の一種である Bluetooth Low Energy (BLE)により双方向にデータを送受信する機能を開発する。

(4) Android スマートフォン/Arduino : 独自の IoT システムの開発

これまでに開発した機能を組み合わせ、図 3-2 のように独自の IoT システムを設計・開発する。Arduino に搭載したセンサによるセンシングの結果を近距離無線により Android スマートフォンへ送信し、Android スマートフォンはセンシングの結果を可視化する。また、Android スマートフォンでは、センシングの結果をもとにアクチュエーションの種類を選択し、その結果を近距離無線により Arduino へ送信する。Arduino は、受信したアクチュエーションの種類をもとにアクチュエータを制御する。

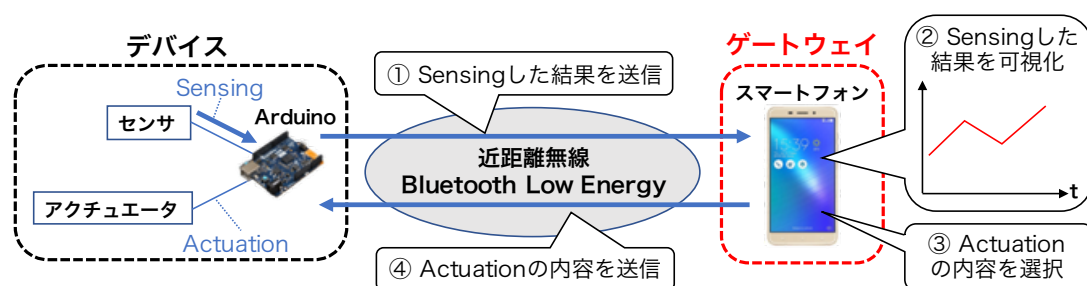


図 3-2. 独自の IoT システムの動作概要

4. Arduino : センシング/アクチュエーションの機能を備えたデバイスの開発

4.1. Arduino とは？

Arduino (アルデュイーノ) とは、マイコン (電化製品などに組み込む用途の小型プロセッサ) と様々な入出力ポートを備えた基盤であり、初心者でも簡単にプログラムが作成できるように統合開発環境 (Arduino IDE) が提供されている。また Arduino は、その設計の全てが公開されているオープンソースハードウェアであり、多くの Arduino 互換機 (図 4-1) が開発されている。これらの Arduino 互換機は、Arduino IDE でプログラムが開発できる。

この実験では、BLE (Bluetooth Low Energy) に対応した互換機である Genuino を利用する。基盤の構成は一般的な Arduino と同じであり、アナログ・デジタル・シリアルといった様々な入出力ポートを備えており、様々なセンサ・アクチュエータを接続して独自のデバイスを簡単に試作することができる。注意点として、Arduino は回路がむき出しになっており壊れやすい機材なので、取り扱いには十分に注意すること。特に、Arduino をノート PC の上など、導電性の高いものの上には絶対に置かないこと。

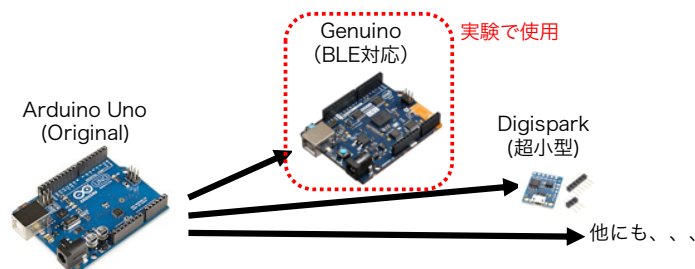


図 4-1. Arduino いろいろ

4.2. Arduino IDE によるデバイスの開発手順

(1) (BYOD の受講生のみ) Arduino IDE のインストール

以下のホームページから Arduino IDE をダウンロードし、インストールする。使用している OS によってダウンロードするファイルが異なるので、注意すること。なお、「Contribute to the Arduino Software」という寄付を求めるページでは、「JUST DOWNLOAD」をクリックすれば良い。

<https://www.arduino.cc/en/Main/Software>

Windows にインストールする場合には、Windows Installer をダウンロードして起動する。途中でアクセス権限や USB ドライバなどについてウィンドウが表示されるが、許可してインストールを進める。

Mac OS にインストールする場合には、ダウンロードしたファイルをアプリケーションフォルダに格納する。

(参考・Windows) <https://poche.xsrv.jp/iot-gym/2019/03/01/001/>

(参考・Mac OS) <https://poche.xsrv.jp/iot-gym/2019/03/01/002/>

(2) Arduino IDE の起動

デスクトップ上 (BYOD の場合には、インストール先に) に配置されている Arduino IDE のアイコンをダブルクリック。図 4-2 のようなウィンドウ (1 回生で利用した Processing とほぼ同じ) が表示される。



図 4-2. Arduino IDE

(3) アプリケーションのサンプルを実行


1. Arduino と PC を USB ケーブルで接続。
2. 上部メニューの[ツール]→[ボード]→[ボードマネージャ]をクリックし、「タイプ」の欄で「**Arduino Certified**」を選択し、「**Inter Curie Boards ...**」の項目を選択して「インストール」をクリック（**自身の PC を利用する場合は、初回のみ。ユーザ ID やパスワードの入力を求めるウィンドウが表示された場合には、キャンセルする。**）。
3. 上部メニューの[ファイル]→[スケッチ例]→[01.Basics]→[Blink]をクリックして、サンプルプログラムを開く。
4. サンプルの上書きを避けるために、上部メニューの[ファイル]→[名前をつけて保存]をクリックし、プログラムに名前（例: **test001**）をつけて保存（USB メモリ内のフォルダ（例: [USB メモリのパス]/**Arduino/test001**）に保存すると、バックアップの手間が省ける）。
5. 上部メニューの[ツール]→[ボード]→[**Arduino/Genuino 101**]を選択（「**Arduino/Genuino Uno**」と間違えないように注意。また、新しいバージョンの **Arduino IDE** では、[ツール]→[ボード]→[**Intel Curie (32-bit) Boards**]→[**Arduino/Genuino 101**]）。
6. 上部メニューの[ツール]→[ポート]→[**COM* (Arduino/Genuino 101)**]を選択（*****の部分は PC によって異なる。**Arduino** を PC に接続した際に追加されるポートを選択）。
7. 画面上部の「」をクリック。
8. 画面下部のログに「**SUCCESS: Sketch will execute in about 5 seconds.**」と表示されれば、プログラムの書き込みが正常に完了。図 4-3 のように、基盤上の LED が 1 秒ごとに点滅していれば、プログラムが正しく書き込めている。



図 4-3. サンプルプログラムの実行

4.3. プログラムの基本的な構成

4.2 節で Arduino に書き込んだプログラムのソースコードを、リスト 4-1 に示す。リスト 4-1 からわかるように、プログラムは関数「**setup**」と関数「**loop**」に分けられている。IoT システムのデバイスの基本的な機能は、定期的にセンサーから値を読み取り(センシング)、その値に応じて適切なアクチュエータを制御(アクチュエーション)することである。そのため Arduino のプログラムでは、(2)の **loop** 関数の内部で行うセンシング・アクチュエーションの処理を設計することが基本となる。

なお、Arduino IDE は Processing を参考に開発されたツールであり、Processing と同じような条件分岐や繰り返しの構文が利用できる(ほぼ C 言語と同じ)。

(1) 関数 setup

Arduino が起動した後に、一度だけ実行される処理を記述する部分であり、入出力ポートの初期化などの処理を行う(Processing における **setup** と同じ)。

(2) 関数 loop

関数 **setup** の処理が実行された後に、繰り返し実行される処理を記述する部分であり、入力ポートからの値の読み取りや、出力ポートへの値の書き込みなどの処理を行う。(Processing における **draw** と同じ)。

リスト 4-1. Arduino 上の LED を点滅させるプログラム

```
void setup() {  
  // initialize digital pin LED_BUILTIN as an output.  
  pinMode(LED_BUILTIN, OUTPUT);  
}  
  
// the loop function runs over and over again forever  
void loop() {  
  digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)  
  delay(1000);                     // wait for a second  
  digitalWrite(LED_BUILTIN, LOW);  // turn the LED off by making the voltage LOW  
  delay(1000);                     // wait for a second  
}
```

リスト 4-1 のプログラムで利用している主な関数の役割は以下の通りである。まずは(3)の **delay** の引数を様々な値に変更し、LED の点滅がどのように変化するか確認してみると良い。

(1) pinMode(pin, mode)

指定された入出力ポートの振る舞い(入力 or 出力)を設定する関数。

pin: 設定の対象となる入出力ポートの番号(「LED_BUILTIN」は、Arduino 上の LED のポートの番号を示すマクロ名)。

mode: 入力(INPUT)と出力(OUTPUT)のどちらの設定にするか指定。

(2) digitalWrite(pin, value)

指定された入出力ポートにデジタル信号(HIGH or LOW)を書き込む関数。関数 pinMode により OUTPUT に設定された入出力ポートのみが対象。

pin: デジタル信号を書き込む入出力ポートの番号。

mode: HIGH と LOW のどちらを書き込むか指定。

(3) delay(ms)

指定された時間だけプログラムを停止する関数。

ms: プログラムを停止する時間(単位はミリ秒)。

上記以外の関数については、以下のようなリファレンスや逆引きを参考にとすると良い。

(参考) Arduino 日本語リファレンス, <http://www.musashinodenpa.com/arduino/ref/>

(参考) 逆引き Arduino, https://garretlab.web.fc2.com/arduino/reverse_lookup/index.html

4.4. センシング/アクチュエーション機能の開発

Arduino を中心として、センシングとアクチュエーションを行う簡単なデバイスを試作する。センサーやアクチュエータは、一般的には入出力ポートとセンサー/アクチュエータを導線で接続するが、本実験では簡単化のために、図 4-4 の「Grove Arduino スターターキット」を利用する。このキットを Arduino に被せて接続することで、付属のケーブルを用いて簡単にセンサやアクチュエータを拡張することができる。

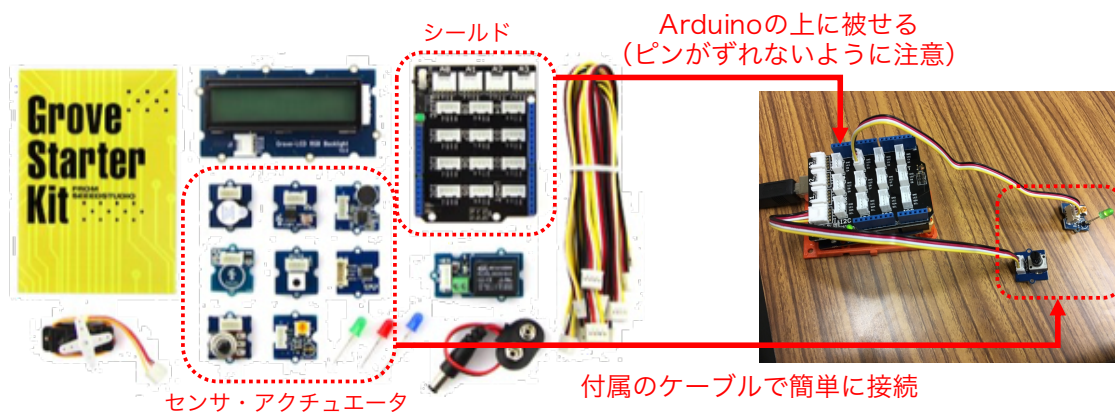


図 4-4. Grove Arduino スターターキット

なお、接続するセンサー・アクチュエータによっては、シールドの **A0** の近くのスイッチを適切な方 (**3V3 or 5V**) に切り替えなければならない点に注意が必要となる。センサ・アクチュエータが正しく動作しない際には、スイッチの切り替えも試みること（基本的には、**5V** とすれば問題ない）。

また、このキットには以下の表 4-1 のように様々なセンサーが同梱されているが、これ以外にも様々なセンサーが販売されている。

表 4-1. Grove Arduino スターターキットに同梱されているセンサー・アクチュエータ

名称	機能
Sound Sensor	音の強さを計測するセンサー
Touch Sensor	指によるタッチを検知するセンサー
Rotary Angle Sensor	ダイヤルの回転角を計測するセンサー
Temperature Sensor	温度を計測するセンサー
Light Sensor	明るさを計測するセンサー
Button (Sensor)	ボタンの押し込みを検知するセンサー
Relay (Actuator)	回路の接続/切断を切り替えるスイッチ
Buzzer (Actuator)	音を出すブザー
LED (Actuator)	光を出す LED
Mini Servo (Actuator)	サーボモーター

これらのセンサーとアクチュエータの中で、まずはセンサーとして「**Rotary Angle Sensor**」、アクチュエータとして「**LED**」を使用し、図 4-5 のような簡単なデバイスを作成する。このデバイスは、**Rotary Angle Sensor** のダイヤルをひねると **LED** の明るさが変わる。

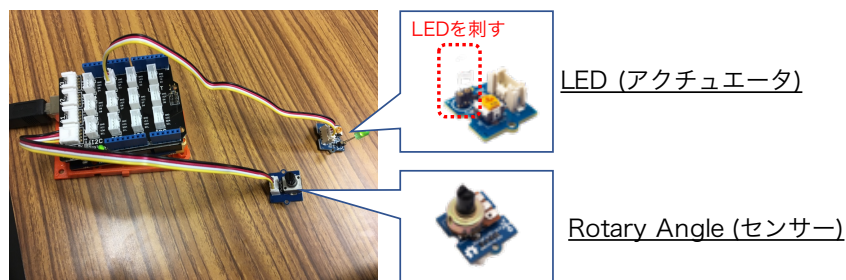


図 4-5. Rotary Angle Sensor と LED を用いたデバイス

デバイスを作成するために、**Rotary Angle Sensor** をシールドの「**A0**」のポートに接続し、**LED** をシールドの「**3**」のポートに接続する。次に、**Arduino IDE** を起動して、上部メニューの[ファイル]→[新規ファイル]をクリックし、新しいソースファイルに名前(例: **test-002**)を付けて保存する。ソースファイルには以下のリスト 4-2 のようなプログラムを記述する。

リスト 4-2. Rotary Angle Sensor と LED で構成されたデバイスのプログラム

```
const int pinrotary = A0;
const int pinled = 3;

void setup() {
  Serial.begin(9600); // シリアルモニタと 9600bps で通信するように設定

  pinMode(pinrotary, INPUT); // Rotary Angle Sensor を接続したポートを入力に設定
  pinMode(pinled, 3);        // LED を接続したポートを出力に設定
}

void loop() {
  int value = analogRead(pinrotary); // Rotary Angle Sensor から値を読み取り
  Serial.println(value);             // センサーから読み取った値をシリアルモニタにログ表示

  analogWrite(pinled, value/4); // センサーから読み取った値の 1/4 を LED の明るさに設定

  delay(100); // 100 ミリ秒待機
}
```

リスト 4-2 のプログラムで利用している主な関数の役割は以下の通りである。なお、4.3 節で紹介した関数は省略する。

(1) Serial.begin(speed)

シリアル通信のデータ転送レートを **bps** で指定して、シリアルモニタを利用する準備をする。

speed: シリアル通信のデータ転送レート

(2) int analogRead(pin)

指定された入出力ポートからアナログ値 (0～1023) を入力し、その値を返り値とする。アナログ値を出力するセンサーは、計測した値に対応した電圧を入出力ポートにかける。この関数は、この入出力ポートにかかっている電圧を 1024 段階の数値に変換している。

pin: アナログ信号を読み込む入出力ポートの番号。

(3) analogWrite(pin, value)

指定された入出力ポートへアナログ値を出力する。アナログ値を入力とするアクチュエータは、入出力ポートにかかった電圧に対応した挙動をとる (例: 電圧が大きい→LED が明るい)。この関数では、この入出力ポートにかかる電圧を 256 段階 (0～255) で指定する。

pin: アナログ信号を書き込む入出力ポートの番号。

value: 入出力ポートに書き込むアナログ値

(4) Serial.println(data, format)

シリアルモニタに文字列やデータを表示する関数であり、Processing における **println** と同様の関数である。リスト 4-2 のプログラムを実行した場合には、図 4-6 のようにセンサーから取得した値がシリアルモニタに表示される。この画面を表示するには、Arduino IDE のシリアルモニターを起動するアイコンをクリックする (図 4-2)。(注: この機能を利用できるのは、自分の PC で作業をしている BYOD の受講者のみです。実験室の PC では利用できません。)



図 4-6. シリアルモニタへセンサーから取得した値を表示

4.5. 課題

4 章については、以下の 3 つの課題に取り組むこと。

- 【課題4-1】 4.3 節の手順に従い、Arduino 上に設置されている LED を点滅させるデバイスを作成すること。また、関数 `delay` の引数を変更した時に、点滅のパターンがどのように変化するか調査すること。
- 【課題4-2】 4.4 節の手順に従い、Rotary Angle Sensor のダイヤルをひねると LED の明るさが変わるデバイスを作成すること。
- 【課題4-3】 4.4 節で利用したものとは別のセンサーとアクチュエータを選択して、独自のデバイスを作成すること。また、どのような目的を意図して作成したデバイスか説明すること(例えば、Light Sensor と LED を利用して、周辺が暗くなると自動的に明かりをつけるデバイスを設計した、など)。センサーやアクチュエータの使用方法については、[4-1] の URL を参考にすること。(本テキストで紹介している以外のセンサーやアクチュエータを利用したい場合には、[4-2]のホームページから選んで教えてください。珍しいセンサーを利用してデバイスを作成するほど高評価(加点)です。)

(参考) Grove Arduino スターターキットのサンプルプログラム,

https://github.com/Seeed-Studio/Sketchbook_Starter_Kit_for_Arduino

(参考) Seeed Wiki (右メニューの Sensor や Actuator を選択), <http://wiki.seeedstudio.com/>

【課題 4-1】～【課題 4-2】については、課題が完成したら「1) 理解度を確認するための TA による口頭試問」を受け、「2) manaba+R へのソースコードの提出」を行うこと。

【課題 4-3】については、課題が完成したら「1) 理解度を確認するための TA による口頭試問」を受け、「2) manaba+R へのレポートの提出」を行うこと。

5. Android スマートフォン：センシング結果を可視化するアプリケーションの開発

5.1. Android とは？

Android とは Google 社が開発したスマートフォン向けのオペレーティングシステムである。プログラミング演習で使った Linux がベースとなっており、タッチスクリーンやセンサ (GPS、加速度、ジャイロなど) といったスマートフォン独自の機能が容易に利用できるように設計されている。また、Android 用のアプリケーションは Android Studio という統合開発環境を提供するツール (プログラミング演習で使った Eclipse のようなもの) により開発でき、このツールで開発したアプリケーションは、Android を搭載したどのスマートフォンにもインストールして実行することができる。

5.2. Android Studio によるアプリケーションの開発手順

(1) (BYOD の受講生のみ) Android Studio のインストール

以下のホームページから Android Studio をダウンロードし、インストールする。

<https://developer.android.com/studio?hl=ja>

基本的にはダウンロードしたファイルを起動し、指示に従ってインストールを進めれば良い。以下のホームページが参考になる。注意点としては、「SDK 等の追加インストール」の作業では、「Android 6.0 (Marshmallow)」も追加で選択してインストールすること。

(参考・Windows) <https://akira-watson.com/android/adt-windows.html>

(参考・Mac OS) <https://akira-watson.com/android/adt-mac.html>

【オンライン受講であり、Android スマートフォンを所有していない場合】

以下のホームページを参考に Android スマートフォンのエミュレータをインストールする。

<https://qiita.com/yacchi1123/items/5849df8965de19818617>

ホームページの「1-3. ADV Manager の設定」に従い、ADV Manager を起動する。「Your Virtual Devices」のウィンドウの左下にある「+ Create Virtual Device」をクリックし、エミュレータを作成する。課題 5 については、ホームページに書かれているように「Nexus 5」で問題ないと思われる。エミュレータを作成したら、必ず再生ボタンを押してエミュレータを起動しておくこと。

(2) Android Studio の起動

デスクトップ上に配置されている Android Studio のアイコンをダブルクリック。

(3) 新しいプロジェクトの作成 (2 回目以降は不要)

【注: BYOD で新しいバージョンの Android Studio を利用する場合】

以降の 1～5 で説明する手順とは異なる。まず、「Select a Project Template」で「Empty Activity」を選択して「Next」をクリックする。次に「Configure Your Project」と表示された画面で、「Application name」にこれから作成するアプリケーション名を入力し、「Save location」ではアプリケーションの開発に関連するファイル群を保存するフォルダを選択する。また、「Language」では Java を選択し、最後に「Finish」をクリックする。

【注: 以降は、実験用のコンピュータを利用する場合】

1. 「Welcome to Android Studio」の画面で、「Start a new Android project」をクリック。
2. 「Create Android Project」と表示された画面では、「Application name」にこれから作成するアプリケーション名 (例: test_app-001)を入力し、「Project location」ではアプリケーションの開発に関連するファイル群を保存するフォルダを選択。最後に「Next」をクリック。
3. 「Target Android Devices」と表示された画面では、「Phone and Tablet」をチェックし、「API 23: Android 6.0 (Marshmallow)」を選択して「Next」をクリック。
4. 「Add an Activity to Mobile」と表示された画面では、「Empty Activity」を選択して「Next」をクリック。
5. 「Configure Activity」と表示された画面では、「Finish」をクリック。
6. 「Tip of the Day」などのウィンドウが表示された場合には、「Close」をクリック。

(4) 開発前の事前作業

1. 図 5-1 のような画面の中で、上部メニューの[View]→[Toolbar]をクリックし、メニューの下に開発用の機能のボタンを表示(BYOD の場合には不要)。
2. 念のため、上部メニューの[Build]→[Rebuild Project]をクリックし、プロジェクト全体を再ビルド。

(5) 画面のレイアウトを設計

1. 図 5-1 のような画面の中で、プロジェクトに関連するファイルが一覧表示されている左側ビューで、[app]→[res]→[layout]→[activity_main.xml]をダブルクリック
2. 開発画面が表示されている右側のビューで[activity_main.xml]のタブを選択し、アプリケーションの画面に必要な部品を追加(詳しくは後述)。

(6) スマートフォンの機能を利用する権限を設定

1. 左側のビューで、[app]→[manifests]→[AndroidManifest.xml]をダブルクリック
2. 右側のビューで[AndroidManifest.xml]を選択し、表示された設定ファイルにアプリケーションの実行に必要なスマートフォンの機能を利用するための権限を設定(詳しくは後述)。

(7) ソースコードを編集

1. 左側のビューで、[app]→[java]→[プロジェクト名(com...test_app-001)]→[MainActivity]をダブルクリック。
2. 右側のビューで[MainActivity.java]を選択し、表示されたプログラムのソースコードを編集(ファイルを保存すると自動的にコンパイルされる。詳しくは後述)。

(8) アプリケーションをスマートフォンにインストール

1. スマートフォンを USB ケーブルで PC に接続し、上部メニューの「▶」のボタンをクリック。
2. 「Select Deployment Target」の画面で、アプリケーションをインストールするスマートフォン(ASUS ASUS_X00DDA (Android 6.0.1, API23))を選択して「OK」をクリック。
3. しばらくすると、スマートフォン上でアプリケーションが起動し、「Hello World!」と書かれた画面が表示。

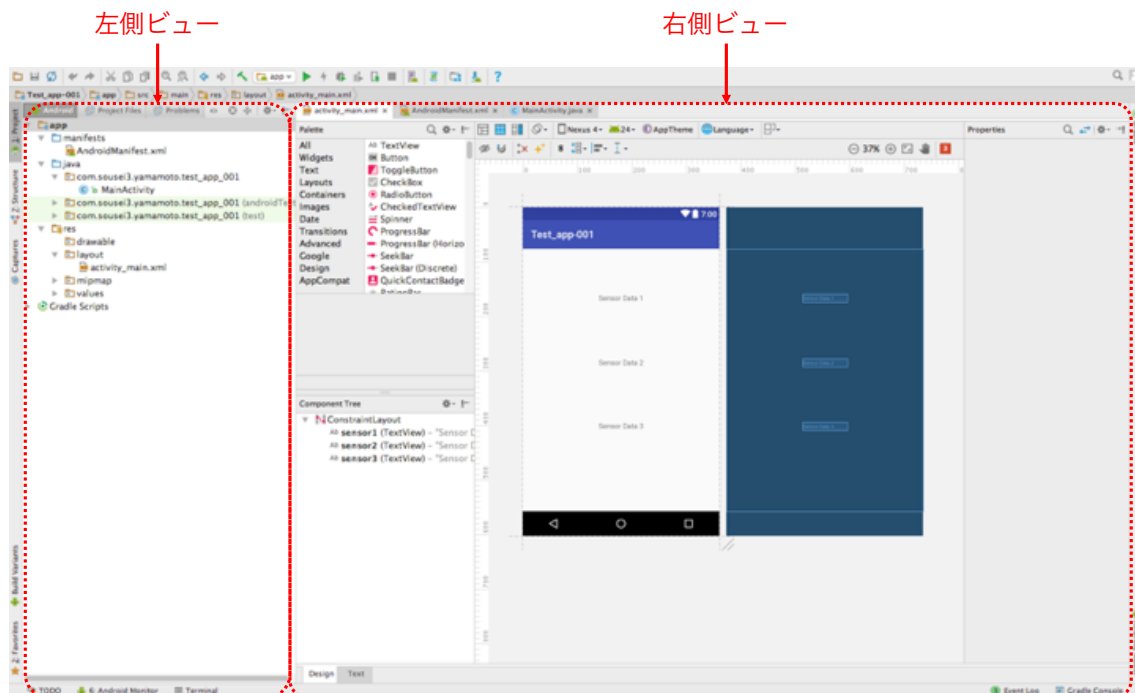


図 5-1. センサからの情報をテキスト表示するアプリケーションの実行例

(9) アプリケーションの開発を終了（週の最後）

1. 上部メニューの[File]→[Export to Zip File]をクリックし、プロジェクト全体を圧縮した Zip ファイルを作成して、USB や OneDrive にコピー(**BYOD** の場合には不要)。

(10) 過去に作成したプロジェクトの読み込み（初回は不要。**BYOD** の場合には不要）

1. 「Welcome to Android Studio」の画面で、「Open an existing Android Studio project」をクリック。
2. 過去に作成したプロジェクトのファイルが保存されているフォルダ(例: (8)で作成した Zip ファイルを解凍したフォルダ)を選択して「OK」をクリック。

5.3. Android スマートフォンに搭載されているセンサからの情報取得

5.2 節で作成したアプリケーションの雛形を用いて、まずは図 5-2 のように、スマートフォンに搭載されているセンサ(加速度、ジャイロなど)から情報を収集して画面にテキスト表示する簡単なアプリケーションを開発する。5.2 節で説明した通り、「画面のレイアウトを設計」、「スマートフォンの機能を利用する権限を設定」、「ソースコードを編集」の 3 つの手順でアプリケーションを開発する。

なお、エミュレータを利用している場合には、エミュレータの右に表示される縦長のメニューバーの「…」をクリックし、「Virtual sensors」の項目をクリックする。表示された **Android** スマートフォンの **3D** モデルの角度を変えることで、加速度センサやジャイロスコープの値を変更できる。また、各センサの値を直接設定することも可能である。

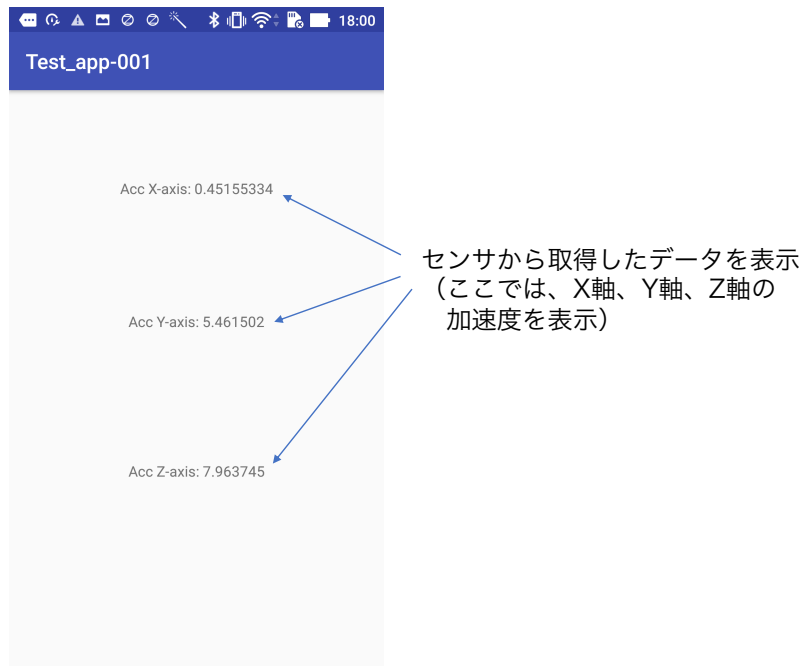


図 5-2. センサからの情報をテキスト表示するアプリケーションの実行例

5.3.1. 画面のレイアウトを設計

図 5-2 では、スマートフォンに搭載されている加速度センサから値を取得して、X 軸、Y 軸、Z 軸の値をそれぞれ表示している。5.2 節で作成したアプリケーションではテキストの表示欄が 1 つだけであるため、あと 2 つ、表示欄を増やす必要がある。5.2 節の(4)で説明したように、右側のビューで [activity_main.xml] のタブを選択し、以下の手順で表示欄を追加する。

(1) レイアウトにテキスト表示用の項目を追加

1. 右側ビューの左上の「Palette」から[All]→[TextView]を選択し、左下の「Component Tree」へドラッグ&ドロップ(BYOD の場合には、[Text]→[TextView]を選択)。
2. もう1つ表示欄を追加するために、1 を繰り返す

(2) 各表示欄の場所を調整

右側ビューの中央の GUI を利用して、各表示欄の表示場所を調整する。各表示欄の位置はマウス操作で変更できる。また、表示欄の上下左右の「○」をマウスのドラッグ&ドロップで接続することで、表示欄の上下関係を固定できる(ここでは、あまりレイアウトには拘らなくて良い)。

(3) プログラムからアクセスできるように、各表示欄に ID を設定

1. 各表示欄をクリックし、右側ビューの右側に表示される「Attributes」の中の「ID」を変更。今回は、各表示欄の ID を、それぞれ sensor1, sensor2, sensor3 と設定。
2. 「Properties」の中の「text」を変更することで、アプリケーション起動時に表示欄に表示される文字列を変更できる。今回は変更しなくても良い(BYOD の場合には、「Common Attributes」の「text」を変更)。

5.3.2. スマートフォンの機能を利用する権限を設定

5.2 節の(5)で説明したように、右側ビューで[AndroidManifest.xml]のタブを選択することで、スマートフォンの機能を利用する権限を設定するファイルである **AndroidManifest.xml** を編集できる。

スマートフォンに搭載されているセンサーの多くは、特に権限を設定しなくても値を取得することができる。権限の設定が必要となるセンサーの 1 つとして現在地を取得する **GPS** があるが、本実験では利用しなくても良い。

アプリケーションを実行している際にスマートフォンを回転させると、スマートフォンの向きに応じて表示が縦方向⇄横方向に自動的に切り替わる。これが煩わしい場合には、**AndroidManifest.xml** の記述をリスト 5-1 のように変更することで、表示の自動切り替えを抑制できる。ここでは画面の表示をスマートフォンの縦方向で固定しているが、“portrait”を“landscape”に変更することで、横方向に固定することも可能である。

リスト 5-1. 表示の自動切り替えの抑制 (AndroidManifest.xml の変更内容)

```
(変更前)
<activity android:name=".MainActivity">

(変更後)
<activity
    android:name=".MainActivity"
    android:screenOrientation="portrait">
```

5.3.3. ソースコードを編集

Android スマートフォン用のアプリケーションは、代表的なオブジェクト指向型の言語となる **Java** で開発することになる。本実験では **Java** によるプログラミングについて詳しい説明は行わないため、必要に応じて自習すること。

5.2 節の(6)で説明したように、右側ビューで[MainActivity.java]のタブを選択することで、アプリケーションのソースコードを編集することができる。プロジェクトを作成した段階では、ソースコードはリスト 5-2 のようになっている。「import ...」の左に設置されている「+」をクリックすると、「import」で始まる全ての行が表示される。

リスト 5-2. アプリケーションのソースコードの初期状態 (MainActivity.java)

```
package com.sousei3.yamamoto.test_app_001;

import android.support.v7.app.AppCompatActivity;
(BYOD の場合、androidx.appcompat.app.AppCompatActivity となっている)
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

(1) 「import ...」

C 言語における「include<...>」に相当し、プログラムで利用する機能(クラス)を読み込むために利用する。この実験では、センサーからの値の取得や画面表示のために、以下のリスト 5-3 のようにクラスを読み込む行を追加する必要がある。

リスト 5-3. プログラムへの機能(クラス)の追加 (MainActivity.java)

```
import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.view.WindowManager;
import android.widget.TextView;
import java.util.List;
```

(2) 「public class MainActivity extends AppCompatActivity {...}」

Android では、アプリケーションの 1 つの画面ごとに、1 つのクラスを作成する必要がある。「MainActivity」が、センサーから取得した情報を表示する画面に対応するクラスとなる。このクラスの中に、センサーからのデータの取得や画面表示などの機能をプログラミングすることになる。

まずは、この「MainActivity」のクラスに、スマートフォンに搭載されているセンサーから情報を取得する基本的な関数を取り扱うことができるように、以下のリスト 5-4 のように変更を加える必要がある。

リスト 5-4. クラスへのセンサーを取り扱うための機能の追加 (MainActivity.java)

(変更前)

```
public class MainActivity extends AppCompatActivity {
```

(変更後)

```
public class MainActivity extends AppCompatActivity implements SensorEventListener{
```

なお、上記を変更すると、後述する「onResume()」、「onPause()」、「onSensorChanged()」、「onAccuracyChanged()」を記述するまで赤波線が引かれるが、現時点では無視して問題ない。

(3) クラス全体で利用するメンバ変数の定義

リスト 5-4 の次の行から、クラス内の全ての関数(メソッド)から参照できる変数(メンバ変数)を定義する。この実験では、以下のリスト 5-5 のメンバ変数を定義して利用する。

リスト 5-5. クラスへのメンバ変数の追加 (MainActivity.java)

```
// センサーマネージャを定義
private SensorManager manager;

// 画面の各表示欄を制御するための変数（今回は3個、必要に応じて増やす）
private TextView sensor1, sensor2, sensor3;

// センサーから届いた値を格納する配列を定義
private float[] values = new float[3];
```

(4) コールバック関数の定義

アプリケーションの起動・開始・停止など、アプリケーションの状態が変化するたびに、アプリケーションは **Android** から特定のイベントを受信する。加えて、スマートフォンに搭載されているセンサーからの情報の取得も、**Android** からアプリケーションに届くイベントの 1 つとなる。

アプリケーションがイベントを受信すると、そのイベントに対応する関数(コールバック関数)に記述した処理が実行される。つまり **Android** におけるプログラミングとは、アプリケーションが取り扱うイベントに対応するコールバック関数の作成が中心となる。

この実験で作成するアプリケーションでは図 5-3 のようなイベントが発生し、これらのイベントに対応するリスト 5-6 のコールバック関数を作成する必要がある。各コールバック関数で行う処理については、以降で説明する。

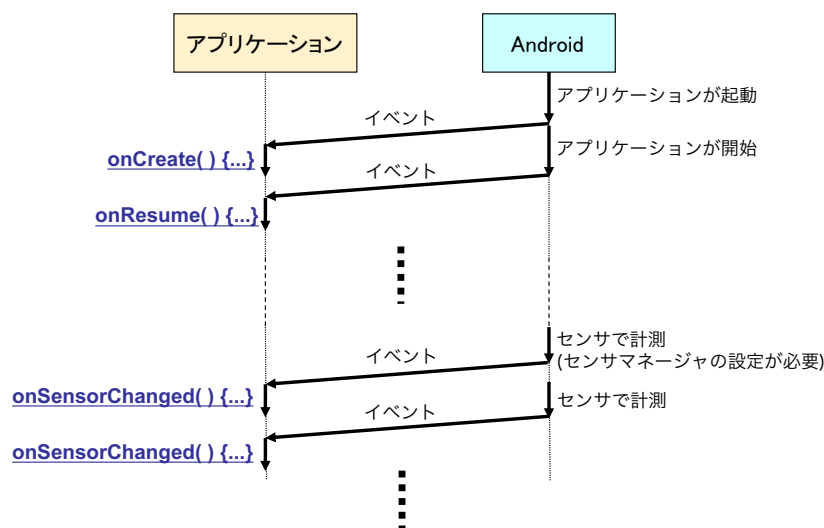


図 5-3. Android から届くイベントと対応するコールバック関数

リスト 5-6. クラスへのコールバック関数の追加 (MainActivity.java)

```
// アプリケーション起動時に呼ばれるコールバック関数
@Override
protected void onCreate(Bundle savedInstanceState) {...}

// アプリケーション開始時に呼ばれるコールバック関数
@Override
protected void onResume(){...}

// アプリケーション一時停止時に呼ばれるコールバック関数
@Override
protected void onPause(){...}

// センサーイベント受信時に呼ばれるコールバック関数
public void onSensorChanged(SensorEvent event){...}

// センサーの精度の変更時に呼ばれるコールバック関数(今回は何もしない)
public void onAccuracyChanged(Sensor sensor, int accuracy){}
```

「onResume()」と「onPause()」は、{...}の中に「super.onResume();」、「super.onPause();」と記述すると、赤波線が消える。詳しくは後述。

(5) 「protected void onCreate(Bundle savedInstanceState) {...}」

アプリケーション起動時に呼び出されるコールバック関数「onCreate」では、主にアプリケーションの初期化に関連する処理を行う。以下のリスト 5-7 のように、画面の表示欄を制御するための変数や、センサーを制御する変数の初期化を行う処理を追加する。

ここで、関数「findViewById」の引数である「R.id.[xxxx]」の「xxxx」の部分は、5.3.1 節の(3)で設定した、各表示欄に対応する ID (sensor1, sensor2, sensor3)となる。

リスト 5-7. onCreate へ追加する処理 (MainActivity.java)

```
// 画面の各表示欄を制御するための変数の初期化
sensor1 = (TextView)findViewById(R.id.sensor1);
sensor2 = (TextView)findViewById(R.id.sensor2);
sensor3 = (TextView)findViewById(R.id.sensor3);

// センサーを制御するための変数の初期化
manager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

(6) 「protected void onResume() {...}」

アプリケーションが起動した後、開始時に呼び出されるコールバック関数「onResume」では、以下のリスト 5-8 のように、主に情報を取得するセンサーの設定と、センサーからの情報の取得を開始する処理を行う。

リスト 5-8. onResume で実行する処理 (MainActivity.java)

```
super.onResume();

// 情報を取得するセンサーの設定 (今回は加速度センサを取得)
List<Sensor> sensors = manager.getSensorList(Sensor.TYPE_ACCELEROMETER);
Sensor sensor = sensors.get(0);

// センサーからの情報の取得を開始
manager.registerListener(this, sensor, SensorManager.SENSOR_DELAY_UI);
```

リスト 5-8 の「manager.getSensorList」は、引数として指定した種類のセンサーを取得する関数である。今回は加速度センサーに対応する「Sensor.TYPE_ACCELEROMETER」の定数を指定している。以下の表 5-1 のような定数を指定することで、他のセンサーを利用することができる。

表 5-1. スマートフォンで利用できる代表的なセンサー

定数	意味
Sensor.TYPE_ACCELEROMETER	加速度センサー
Sensor.TYPE_GYROSCOPE	ジャイロスコープ
Sensor.TYPE_MAGNETIC_FIELD	地磁気センサー (コンパス)
Sensor.TYPE_LIGHT	照度センサー
Sensor.TYPE_PROXIMITY	接近センサー

(7) 「protected void onPause() {...}」

ホーム画面に戻った時など、アプリケーションが停止した時に呼び出されるコールバック関数「onPause」では、以下のリスト 5-9 のように、主にセンサーからの情報の取得を停止する処理を行う。

リスト 5-9. onPause で実行する処理 (MainActivity.java)

```
super.onPause();  
  
// センサのリスナー登録解除  
manager.unregisterListener(this);
```

(8) 「protected void onSensorChanged(SensorEvent event) {...}」

アプリケーションがスマートフォンのセンサーから情報を取得した時に呼び出されるコールバック関数「onSensorChanged」では、以下のリスト 5-10 のように、主にセンサーから取得した情報を各表示欄に表示する処理を行う。

リスト 5-10. onSensorChanged で実行する処理 (MainActivity.java)

```
// 取得した情報が加速度センサーからのものか確認  
if(event.sensor.getType() == Sensor.TYPE_ACCELEROMETER){  
  
    // 受け取った情報を格納用の配列にコピー  
    values = event.values.clone();  
  
    // 受け取った情報を表示欄に表示  
    sensor1.setText("Acc X-axis: " + values[0]);  
    sensor2.setText("Acc Y-axis: " + values[1]);  
    sensor3.setText("Acc Z-axis: " + values[2]);  
}
```

センサーから取得した情報は、関数「onSensorChanged」の引数である「event」に格納されている。この関数には複数のセンサーから情報が届く可能性があるため、まずはどのセンサーから届いた情報か確認する必要がある。そこで、引数「event」のメソッドである「event.sensor.getType()」を実行してセンサーの種類を取得し、目的のセンサー(リスト 10 の場合には加速度センサー)から収集された情報であるか確認している。

加速度センサーの場合は、1 つ目の要素(values[0])に X 軸の加速度、2 つ目の要素(values[1])に Y 軸の加速度、3 つ目の要素(values[2])に Z 軸の加速度が格納される(注: センサーによって要素数は異なる)。センサーから取得したこれらの情報を含む文字列を関数「sensor[1-3].setText」の引数に指定することで、各表示欄の表示内容を変更している。

5.4. グラフ表示機能の開発

5.3 節で作成したセンサーから取得したデータをテキスト表示するアプリケーションをもとに、図 5-4 のように、スマートフォンに搭載されているセンサ(加速度、ジャイロなど)からデータを収集して画面にグラフ表示するアプリケーションを開発する。グラフ表示を行う機能を利用するための前準備を行なった上で、4.2 節で説明した「画面のレイアウトを設計」、「スマートフォンの機能を利用する権限を設定」、「ソースコードを編集」の 3 つの手順でアプリケーションを開発する。

この実験では、外部のライブラリを利用することで、Android のアプリケーションに対して簡単に機能を追加できることを体験する。この実験の開発に取り組む際には、新しくプロジェクト(例: test_app-002)を作成すること。



図 5-4. センサからの情報をグラフ表示するアプリケーションの実行例

5. 4. 1. グラフ表示機能を利用するための前準備

グラフ表示機能を利用するために、この実験では「MPAndroidChart [5-1]」と呼ばれるライブラリを利用する。このライブラリをプロジェクトに組み込むために、左側のビューで[Gradle Scripts]→[build.gradle (Module:app)]をダブルクリックし、右側のビューでライブラリの設定ファイルを開く。この設定ファイルを以下のリスト 5-11 のように編集し、画面の右上に表示される「Sync Now」をクリックすることで、MPAndroidChart をプロジェクトに組み込むことができる(組み込みには時間がかかるため、以降の作業を先に進めておくと良い。エラーが発生する場合には、上部メニューで[Build]→[Rebuild Project]をクリックすると修復することが多い)。

リスト 5-11. MPAndroidChart をプロジェクトへ組み込み (build.gradle の変更内容)

```
(“android{“の次の行に追加)
repositories{
    maven {url “http://jitpack.io”}
}

(“testImplementation ‘junit:junit:4.12.0’”の前の行に追加)
implementation ‘com.github.PhilJay:MPAndroidChart:v3.0.3’
```

(参考) MPAndroidChart, <https://github.com/PhilJay/MPAndroidChart>

5.4.2. 画面のレイアウトを設計

右側ビューで[activity_main.xml]を選択して、アプリケーションの画面のレイアウトを編集する。ここで、[activity_main.xml]の標準の画面ではグラフ表示用の項目は選択できないため、右側ビューの左下のタブを[Design]から[Text](BYOD の場合は[Code])に変更する。5.3 節のアプリケーションを作成した後だと、テキスト表示用の項目に対応する「<TextView … />」というタグが 3 つ並んでいるはずである(1 つのタグが複数行に渡っている)。この 3 つのタグを全て削除し、リスト 5-12 のように、グラフ表示用の項目に対応するタグを追加する。

リスト 5-12. グラフ表示用の項目の追加 (activity_main.xml の変更内容)

```
<com.github.mikephil.charting.charts.LineChart
    android:id="@+id/chart_DynamicMultiLineGraph"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

ここで、「android:id=…」の行がこの項目の ID の設定となり、この ID を参照することで、プログラム本文 (MainActivity.java) からグラフへのデータの追加などが可能となる。

5.4.3. スマートフォンの機能を利用する権限を設定

5.3 節のアプリケーションと同様に、この実験で開発するアプリケーションも、権限の設定が必要となるようなスマートフォンの機能は利用しない。必要に応じて、5.3.2 節ように、画面表示の自動切り替えを抑制する設定を加えると良い。

5.4.4. ソースコードを編集

5.3.3 節で説明したように、右側ビューで[MainActivity.java]のタブを選択することで、アプリケーションのソースコードを編集することができる。ここで、グラフ表示に関するプログラムを一から作成していると非常に時間がかかるため、この実験では雛形となるソースコードを用意している。

manaba+R のコンテンツの「Android:グラフ表示機能の開発」から「MainActivity.java」をダウンロードし、その内容を自身の[MainActivity.java]の「import android.os.Bundle」の行以降にコピーすること。この段階で、アプリケーションをスマートフォンにインストールして実行すると、グラフ画面が表示されるはずである。

このプログラムは、MainActivity クラスのメンバ変数として定義されているリスト 5-13 の変数を初期化・更新することで、任意のデータをグラフ表示できるようになっている。num はグラフに表示するデータの個数であり、雛形のように X 軸、Y 軸、Z 軸の加速度という 3 つのデータを表示する場合には、num には 3 を代入する。また labels と colors には、それぞれのデータの名前と、グラフに表示する線や点の色を指定する。さらに、max と min には、画面の表示するグラフの Y 軸の最

大値と最小値を指定する。最後に、**values** の各要素に代入した値が、動的(500ms 間隔)にグラフへ追加されることになる。

リスト 5-13. グラフ表示に関連するメンバ変数 (MainActivity.java)

```
private int num; // グラフにプロットするデータの数

private String[] labels; // データのラベルを格納する配列
private int[] colors; // グラフにプロットする点の色を格納する配列
private float max, min; // グラフのY 軸の最大値と最小値

private float[] values; // データを格納する配列
```

雛形のプログラムでは **values** には何も値が代入されていないため、グラフに表示されるデータは全て 0 となっている。5.3 節で作成したアプリケーションのソースコードを参考にして、センサーから取得したデータをこの **values** に代入するように雛形のソースコードを修正することで、センサーから取得したデータをリアルタイムにグラフ表示するアプリケーションが実現できる。

5.5. 課題

5 章については、以下の 4 つの課題に取り組むこと。

- 【課題5-1】 5.3 節の手順に従い、スマートフォンの**加速度センサー**から取得したデータをテキスト表示するアプリケーションを作成
- 【課題5-2】 5.3 節で作成したアプリケーションについて、**加速度センサー「以外」**のセンサーから取得したデータを表示するように変更(加速度センサー・ジャイロスコップ以外の珍しいセンサーを利用している場合には加点)
- 【課題5-3】 5.4 節の手順に従い、スマートフォンの**加速度センサー**から取得したデータをグラフ表示するアプリケーションを作成
- 【課題5-4】 5.4 節で作成したアプリケーションについて、**加速度センサー「以外」**のセンサーから取得したデータをグラフ表示するように変更(加速度センサー・ジャイロスコップ以外の珍しいセンサーを利用している場合には加点)

【課題 5-1】～【課題 5-3】については、課題が完成したら「1) 理解度を確認するための TA による口頭試問」を受け、合格したら「2) manaba+R へのソースファイルの提出」を行うこと。

【課題 5-4】については、課題が完成したら「1) 理解度を確認するための TA による口頭試問」を受け、「2) manaba+R へのレポートの提出」を行うこと。

6. Android スマートフォン/Arduino：近距離無線による通信機能の開発

6.1. 近距離無線 Bluetooth Low Energy (BLE)とは？

Bluetooth Low Energy (BLE)とは、近距離(100m 以下)の通信を対象とした無線技術である Bluetooth の一部である低消費電力の通信モードである。従来の Bluetooth よりも消費電力が低いことから、様々なセンサーやアクチュエータを接続するセンサネットワークの構築への利用が注目されており、多くのスマートフォンや PC が BLE に対応している。

BLE 通信を行う機器は、セントラルとペリフェラルの二種類に大別できる。セントラルとは BLE 通信の中心となる機器であり、主にスマートフォン/タブレットや PC が担当する。また、ペリフェラルはセントラルから利用される周辺機器であり、キーボード/マウスやヘッドセットなどが相当する。また、IoT システムにおけるセンサーやアクチュエータもペリフェラルとして機能することが多い。

セントラルとペリフェラルの間で行われる BLE 通信には、ブロードキャスト型とコネクション型の二種類が存在する(図 6-1)。

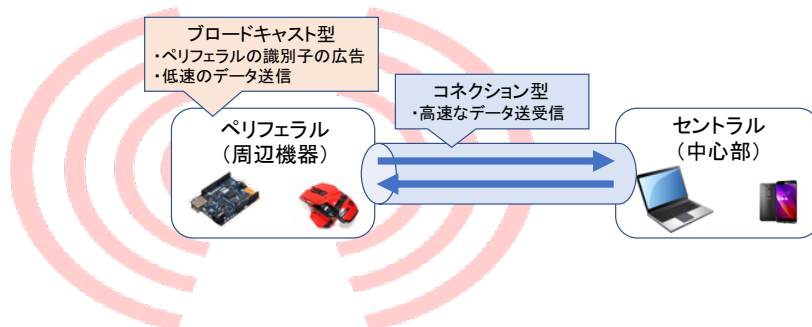


図 6-1. BLE のブロードキャスト型通信とコネクション型通信

(1) ブロードキャスト型通信

主にペリフェラルが送信側となる通信であり、自身の存在を周辺の機器に知ってもらうための広告パケットを送信する際に利用される。この広告パケットには、ペリフェラルを識別するための情報(識別子など)だけでなく、任意のデータを格納することも可能であり、周辺の多くの機器へ一斉にデータを配信するためにも利用できる。また、広告パケットを受信した際には、その送信に利用された電波の受信強度を測定することが可能であり、この受信強度からペリフェラルまでの距離を見積もることができる(Apple 社の iBeacon などの位置情報サービスに利用されている)。

(2) コネクション型通信

広告パケットを受信したセントラルがペリフェラルとの間でコネクションを確立し、一対一でデータを送受信する際に利用される。コネクションの確立に関連するのは以下の 4 つの手続きとなる。

- **Advertise:** ペリフェラルが周辺に広告パケットをブロードキャスト
- **Scan:** セントラルが周辺にどんなペリフェラルが存在するかスキャン
- **Connect:** セントラルが指定したペリフェラルとの間でコネクションを確立
- **Disconnect:** セントラルがペリフェラルとの間のコネクションを破棄

コネクション型の通信はブロードキャスト型よりも高速であり、通信の方向によって以下の 3 種類に分類される(図 6-2)。

- **Read:** セントラルがペリフェラルから値を読み出す
- **Notify:** 値が更新された際に、ペリフェラルからセントラルに対して通知する。
- **Write:** セントラルがペリフェラルへ値を書き込む

また、ペリフェラルは送受信の対象となるデータを管理するために、図 6-2 のようなデータ構造を持っている。サービス(Service)が PC におけるフォルダに相当し、キャラクタリスティック(Characteristic)がファイルに相当し、その内部に値(Value)が格納されている。それぞれのキャラクタリスティックについて、Read/Notify/Write のどの通信に対応させるか指定することができる。

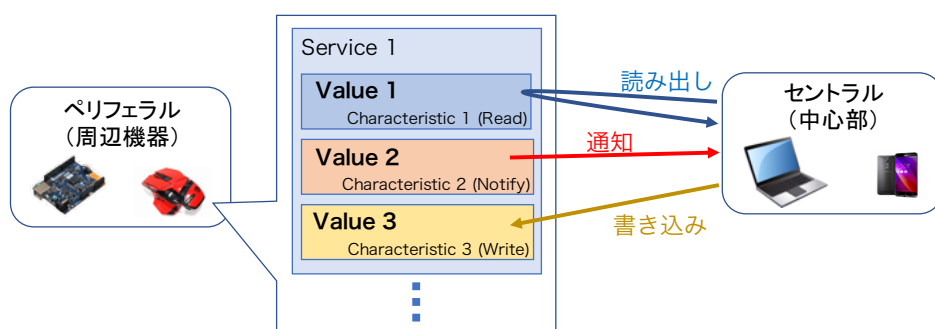


図 6-2. コネクション型通信における 3 種類の通信とペリフェラルのデータ構造

以降の節では、BLE 通信によりデータ送受信を行う、ペリフェラルとセントラルの簡単な機能をそれぞれ試作する。

6. 2. Arduino: ペリフェラル側の開発

セントラルに対して自身の識別子を含む広告パケットをブロードキャスト型通信で送信し、セントラルとの間でコネクションが確立された場合には、セントラルに対して整数値(1, 2, 3, ..., 1000, 1, 2, 3, ...)を順番に送信(Notify)し、セントラルから受信(Write)した整数値をシリアルモニタに表示するペリフェラルの機能(図 6-3)を開発する。

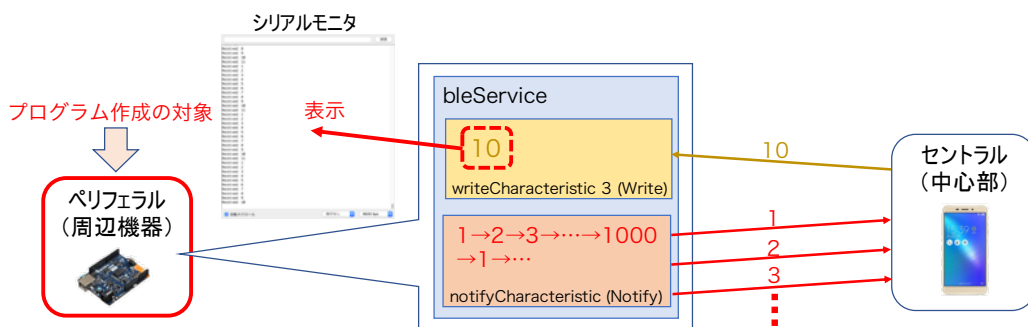


図 6-3. 試作するペリフェラルの機能の概要

Arduino には何もセンサーやアクチュエータを接続せず、Arduino IDE を起動して、上部メニューの[ファイル]→[新規ファイル]をクリックし、新しいソースファイルに名前(例: test-003)を付けて保存する。ソースファイルには、以下のリスト 6-1～6-3 のようなプログラムを記述する。

ここで、プログラムを一から作成していると識別子の入力ミスなどが発生するため、雛形となるソースコードを用意している。manaba+R のコンテンツの「Android スマートフォン/Arduino:近距離無線による通信機能の開発」から「BLEPeripheral.ino」をダウンロードし、その内容を自身のソースファイルにコピーすること。このプログラムの内容について、以降で解説する。この課題に取り組む際には、Android スマートフォンの Bluetooth を On にするのを忘れないこと。

(1) サービスおよびキャラクタリスティックの識別子を設定

セントラルがペリフェラル内のキャラクタリスティックを識別できるように、リスト 6-1 のように識別子を設定してサービスとキャラクタリスティックを生成している。セントラル側は、この識別子をもとにペリフェラルがもつサービスやキャラクタリスティックを検索する。リスト 6-1 のように、このペリフェラルは Write 通信用と Notify 通信用の、2 つのキャラクタリスティックを備えている。

リスト 6-1. サービスとキャラクタリスティックの識別子の設定

```
// Service の作成
BLEService bleService("19B10010-E8F2-537E-4F6C-D104768A1214");

// write(Android -> Arduino への通信)用 Characteristic の作成
BLECharacteristic writeCharacteristic("19B10011-E8F2-537E-4F6C-D104768A1214", BLERead | BLEWrite, 2);
// notify(Arduino -> Android への通信)用 Characteristic の作成
BLECharacteristic notifyCharacteristic("19B10012-E8F2-537E-4F6C-D104768A1214", BLERead | BLENotify, 2);
```

ここで、writeCharacteristic/notifyCharacteristic の第 3 引数は、セントラル/ペリフェラル間で送受信するデータのバイト数を表す。このサンプルプログラムでは、Arduino の int 型(2 バイト)の整数値を送受信することを想定しているため、第 3 引数を「2」としている。例えば、int 型の整数値を一度に 2 つ送受信したい場合には、第 3 引数を「4」とする。

(2) BLE 通信を開始するための初期設定

setup 関数では、リスト 6-2 のように BLE 通信を開始するための初期設定を行なっている。最初に BLE 通信を管理するクラス(BLE)を初期化し、ペリフェラルの名前を設定している。リスト 6-2 では名前を「SecNet2_XXXX」としているが、「XXXX」の部分は自分の学籍番号の下 4 桁とすること。次に、ペリフェラルからブロードキャスト型通信により周辺の機器へ送信する、広告パケットに記録するサービスの識別子を設定し、そのサービスに含まれるキャラクタリスティックを設定している。最後に、各キャラクタリスティックの値を 0 で初期化し、広告パケットの送信を開始している。

リスト 6-2. BLE 通信を開始するための初期設定

```
// BLE の初期化を開始
BLE.begin();

// ペリフェラルの名前を設定 (XXXX は学籍番号の下 4 桁)
BLE.setLocalName("SecNet2_XXXX"); ← 名前は必ず変更すること！！

// ペリフェラルから広告する Service の識別子を設定
BLE.setAdvertisedService(bleService);
```

```
// Service へ Characteristic を追加
bleService.addCharacteristic(writeCharacteristic);
bleService.addCharacteristic(notifyCharacteristic);

// Service を追加
BLE.addService(bleService);

// 広告を開始
BLE.advertise();
```

(3) セントラルとの間のデータ送受信

loop 関数では、リスト 6-3 のようにセントラルとの間でコネクション型通信を行なっている。BLE 通信に関するイベントの処理を開始した後に、セントラルへ送信する値を決定する処理(1~1000 の値を巡回)を行っている。その後、Notify 通信用のキャラクタリスティックに、セントラルへ送信する値を設定している。その後、Write 通信用のキャラクタリスティックから、セントラルから受信した値を読み取り、その値をシリアルモニタに出力している。これらの処理を、500 ミリ秒間隔で実行している。

リスト 6-3. セントラルとの間のデータ送受信

```
// BLE のイベントをポーリング
BLE.poll();

/** セントラルへ送信する値の決定 **/
send_value += 1;
if(send_value > 1000){
    send_value = 1;
}
/*****/

// センサーから取得した値を Android に送信
memcpy(value, (byte *)&send_value, 2);
notifyCharacteristic.setValue(value, 2);

// Android からアクチュエータ制御用の値を受信
memcpy(value, writeCharacteristic.value(), 2);
memcpy((byte *)&recv_value, value, 2);

/** セントラルから受信した値に対する処理 **/
Serial.print("Received: ");
Serial.println(recv_value);
/*****/

delay(500);
```

なお、notifyCharacteristic.setValue はペリフェラルからセントラルへデータを送信する関数であり、第 1 引数は送信するデータが格納されている byte 型の配列であり、第 2 引数はその配列のサイズを指定する。セントラルとペリフェラルの間では、byte 型の配列を介してデータを送受信する。

そのため、`setValue` の関数を呼び出す前に、送信したい整数値が格納されている変数 (`send_value`)の内容を、`memcpy` 関数により `byte` 型の配列にコピーしている。

一方で、`writeCharacteristic.value` はセントラルからペリフェラルへ送信されたデータを取得する関数であり、その返り値は `byte` 型の配列となる。`memcpy` 関数により、`byte` 型の配列を変数 (`recv_value`)にコピーしている。

6.3. Android スマートフォン： セントラル側の開発

指定した識別子を持つペリフェラルとの間でコネクションを確立し、ペリフェラルとの間でコネクションが確立された場合には、ペリフェラルから送信された整数値を受信(Notify)して 5.4 節で作成したアプリケーションと同様にグラフ表示し、受信した値に 1 加算した整数値をペリフェラルに送信(Write)するセントラルの機能(図 6-4)を開発する。また、グラフには広告パケットを受信した際の受信電波強度(RSSI: Received Signal Strength Indication)と、ペリフェラルから受信した整数値を表示する。

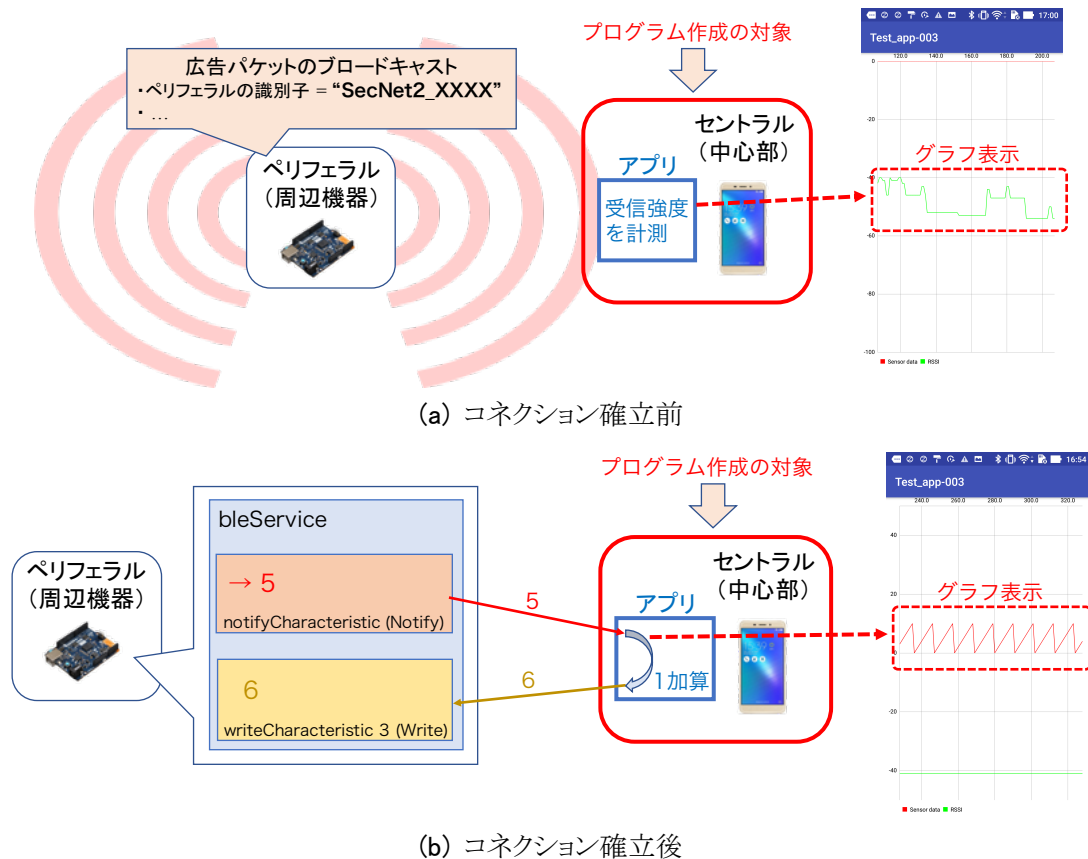


図 6-4. 試作するセントラルの機能の概要

ここでは、5.4.1 節で説明した「グラフ表示を行う機能を利用するための前準備」と、5.4.2 節で説明した「画面のレイアウトを設計」を行った上で、「スマートフォンの機能を利用する権限を設定」と「ソースコードを編集」の 2 つの手順でアプリケーションを開発する。

6.3.1. スマートフォンの機能を利用する権限を設定

スマートフォンが持つ BLE 通信を行う機能を利用するには、その機能を利用するための権限を設定する必要がある。5.3.2 節で説明したように、Android Studio の左側ビューで [app]→[manifests]→[AndroidManifest.xml] をダブルクリックし、右側ビューで [AndroidManifest.xml] を選択する。右側ビューに表示された AndroidManifest.xml の記述に、リスト 6-4 のように追記する(赤文字の部分を追加)。

ここでは、BLE 通信の機能を利用するための権限(BLUETOOTH, BLUETOOTH_ADMIN)に加えて、スマートフォンで計測できる位置情報(GPS など)を利用するための権限(ACCESS_FINE_LOCATION)を設定する必要がある。これは、BLE が iBeacon などの位置情報サービスにも利用できるためである。

リスト 6-4. BLE 通信を利用するための権限の設定 (AndroidManifest.xml の変更内容)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="...">

    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    ...
</manifest>
```

6.3.2. ソースコードを編集

5.3.3 節で説明したように、右側ビューで [MainActivity.java] のタブを選択することで、アプリケーションのソースコードを編集することができる。

ここで、BLE 通信に関するプログラムを一から作成していると非常に時間がかかるため、この実験でも雛形となるソースコードを用意している。manaba+R のコンテンツの「Android スマートフォン/Arduino: 近距離無線による通信機能の開発」から「MainActivity.java」をダウンロードし、その内容を自身の [MainActivity.java] の「import android.os.Bundle」の行以降にコピーすること。この段階で、接続対象となるペリフェラルの名前を 6.2 節で作成した Arduino のプログラムと合わせ、アプリケーションをスマートフォンにインストールして実行すると、ペリフェラルとの間でコネクションを確立してデータを送受信する機能を実現できる。

(1) ペリフェラル/サービス/キャラクタリスティックの識別子を設定

セントラルがペリフェラル内のキャラクタリスティックを識別できるように、リスト 6-5 のように識別子を設定してサービスとキャラクタリスティックを生成している。セントラル側は、この識別子をもとに接続対象となるペリフェラルを特定し、データ送受信の対象となるキャラクタリスティックを参照する。リスト 6-5 ではペリフェラルの名前を「SecNet2_XXXX」としているが、「XXXX」の部分は自分の学籍番号の下 4 桁に変更すること。

リスト 6-5. ペリフェラル/サービス/キャラクタリスティックの識別子を設定

```
// 接続対象となるペリフェラルの名前
private static final String PERIPHERAL_NAME = "SecNet2_XXXX"; ← 名前は必ず変更すること

// 接続対象となるサービスの識別子
private static final String SERVICE_UUID = "19B10010-E8F2-537E-4F6C-D104768A1214";

// 接続対象となるキャラクタリスティックの識別子
private static final String CHAR_WRITE_UUID = "19B10011-E8F2-537E-4F6C-D104768A1214";
private static final String CHAR_NOTIFY_UUID = "19B10012-E8F2-537E-4F6C-D104768A1214";
```

(2) ペリフェラルと接続する/しないを切り替えるフラグを設定

リスト 6-6 では、セントラルが接続対象となるペリフェラルを発見した際に、そのペリフェラルと接続する(flag_connect = 1)か接続しない(flag_connect = 0)かを選択するためのフラグを定義している。ペリフェラルは、セントラルに接続すると広告パケットの送信を停止する。そのため、ペリフェラルから送信されたデータを表示する際にはこのフラグを 1 に、広告パケットを受信した際の受信電波強度を表示する際にはこのフラグを 0 に設定すると良い。

リスト 6-6. ペリフェラルと接続する/しないを切り替えるフラグを設定

```
private int flag_connect = 1;
```

(3) グラフ表示のために利用する各種変数の定義

5.4.4 節のリスト 5-13 と同じように、グラフ表示に関連するメンバ変数を定義し、プログラムを実行したときに呼び出されるコールバック関数(onCreate)では、これらの変数の初期化を行っている。雛形のプログラムでは、グラフに表示する値を格納する配列 values に対して、以下のように値を格納することを想定している。

- values[0] : ペリフェラルから受信したデータを格納
- values[1] : ペリフェラルから広告パケットを受信した際の受信電波強度を格納

この配列に格納した値がグラフ表示されるように、5.4.4 節で作成したプログラムの中から、グラフ表示に関する部分を移植する必要がある。

(4) BLE を利用するための準備

Android のアプリケーションが実行/開始したときに呼び出されるコールバック関数(onCreate/onResume)では、BLE 通信を利用する準備を整え、広告パケットのスキャンを開始するための処理を実行している。

これらの処理を実行した後、ペリフェラルから送信された広告パケットを受信すると、次に説明するコールバック関数(onScanResult())の中で定義した処理が実行される。

(5) ペリフェラルから広告パケットを受信したときの処理

ペリフェラルから広告パケットを受信したときに呼び出されるコールバック関数(`onScanResult`)では、引数 `result` に広告パケットを受信した結果に関する情報が格納されている。本実験のプログラムでは、以下の情報を参照している。

- `result.getDevice().getName()`: 広告パケットを送信したペリフェラルの名前
- `result.getRssi()`: 広告パケットを受信した際の受信電波強度

このペリフェラルの名前を参照し、リスト 6-5 で設定した名前と一致する場合には、広告パケットを受信した際の受信電波強度を取得し、またリスト 6-6 で設定したフラグが 1 である場合には、発見したペリフェラルとの間で接続の確立を試みている。また、広告パケットを受信した際の受信電波強度を、グラフに表示する値を格納する配列の 2 番目の要素(`values[1]`)に格納している。

接続が正しく確立できた場合には、次に説明するコールバック関数(`onConnectionStateChanged`)の中で定義した処理が実行される。

(6) ペリフェラルとの間で接続が確立されたときの処理

ペリフェラルとの間で接続が正しく確立できたときに呼び出されるコールバック関数(`onConnectionStateChanged`)では、ペリフェラルに含まれるサービスの検索を実行している。ここで、サービスが発見できたときに呼び出されるコールバック関数(`onServicesDiscovered`)では、そのサービスがリスト 6-5 で設定した識別子と一致する場合には、同じくリスト 6-5 で設定した識別子と一致するキャラクタリスティックを取得している。このキャラクタリスティックに対して、**Notify** 通信を有効化する手続きなどを実行している。

これらの処理を実行した後、ペリフェラルから **Notify** 通信によりデータを受信すると、次に説明するコールバック関数(`onCharacteristicChanged`)の中で定義した処理が実行される。

(7) ペリフェラルからデータを受信したときの処理

ペリフェラルからデータを受信したときに呼び出されるコールバック関数(`onCharacteristicChanged`)では、まずはペリフェラルから受信したデータに含まれる値を、グラフに表示する値を格納する配列の 1 番目の要素(`values[0]`)に格納している。その後、ペリフェラルから受信した値を入力として、ペリフェラルへ送り返す値を返り値とする関数(`DecideControlParameter`)を実行する。最後に、ペリフェラルへ送り返す値を、**Write** 通信用のキャラクタリスティックに設定している。

6.3. 課題

6 章については、以下の 2 つの課題に取り組むこと。

【課題6-1】 6.2 節と 6.3 節の手順に従い、ペリフェラルとして動作する **Arduino** からセントラルとして動作する **Android** スマートフォンへ整数値(1, 2, 3, ...)を順番に送信し、**Android** スマートフォンは受信した整数値に 1 加算した整数値を **Arduino** へ送信するプログラムを作成すること。まずは、**Android** スマートフォンが **Arduino** とは接続するようにフラグを設定し、**Android** から受信した整数値が正しくグラフ表示できていることを確認すること。

【課題6-2】 (1)で作成したプログラムについて、**Android** スマートフォンが **Arduino** とは接続しないようにフラグを設定し、広告パケットを受信したときの受信電波強度を継続して計測する実験を行うこと。さらに、**Android** スマートフォンと **Arduino** の間の距離が、受信電波強度に与える影響を実験結果としてまとめ、**BLE** が位置情報サービスとして利用できるか考察せよ。(レポートで距離と受信電波強度との関係を示すグラフを描いた上で考察できていると高評価(加点)です。)

【課題 6-1】については、課題が完成したら「1) 理解度を確認するための TA による口頭試問」を受け、「2) manaba+R へのソースファイルの提出」を行うこと。

【課題 6-2】については、課題が完成したら「1) 理解度を確認するための TA による口頭試問」を受け、「2) manaba+R へのレポートの提出」を行うこと。

7. Android スマートフォン/Arduino: 独自の IoT システムの開発

4.5 節の課題(3)で作成した装置について、センシングの結果をもとにアクチュエーションの種類を選択する機能を Arduino から Android スマートフォンに移植し、センシング・プロセッシング・アクチュエーションの機能が近距離無線(BLE)を介して連携する独自の IoT システムを開発する。

これを実現するには、センシングの結果を入力として、アクチュエーションの種類を出力とする機能を、Android スマートフォン側の関数「DecideControlParameter」に移植する。例えば、「周辺が暗くなると自動的に明かりをつける」機能の場合には、Light Sensor から取得した値をこの関数の引数に設定し、LED の明るさを決定する値を計算してこの関数の戻り値とする(図 7-1)。

注意点として、6.3.2 節のために用意しているプログラムの雛形では、BLE により送受信されるデータの大きさを 2 バイトとしているため、送受信できる値は 0～65535 の範囲内となる。そのため、関数「DecideControlParameter」の引数と戻り値の範囲も 0～65535 とする必要がある。センサーから取得する値やアクチュエータへ設定する値がこの範囲外となる場合には、この範囲内となるように対処が必要となる。

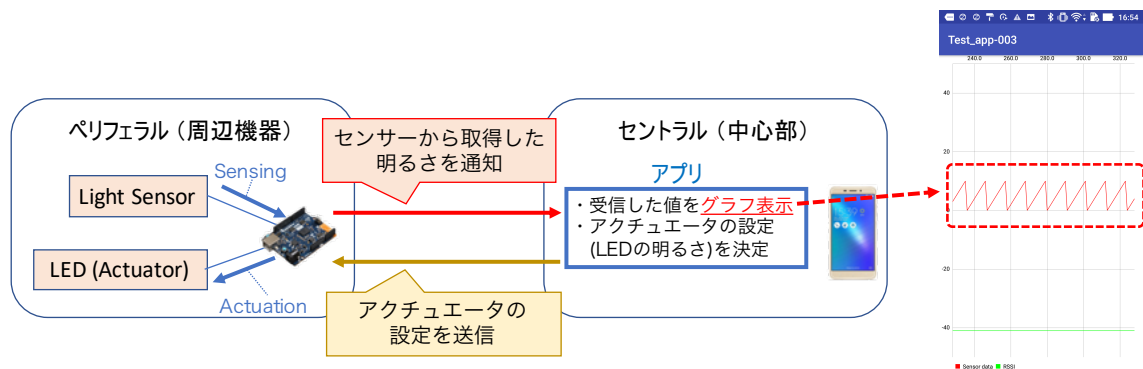


図 7-1. 試作する IoT システムの一例

7 章の課題については、課題が完成したら「2) manaba+R へのレポートの提出」を行うこと。