13 回目　センサーの紹介とその値の取得

今日の講義で学ぶ内容

・センサーについて
・センサーの動作確認
・センサーのデータ取得

センサーについて

§1　携帯端末とセンサー

近年、スマホやタブレットなどの携帯端末の普及が進んでいます。これに伴い、携帯端末に様々なセンサーが搭載されるようになり携帯機器のスマート化が進んでいます。傾きセンサーや方位センサー、ジャイロセンサー、GPS センサー、明るさセンサーなどが代表的なセンサーです。携帯端末で動作するアプリはこれらのセンサーからデータを適切に取得しながら、ユーザに寄り添ったサービスを提供します。

最近では、身に着けて使用するウェラブル端末が人気を集めています。Google グラスやスマートウォッチが良い例です。これらのウェラブル端末もセンサーを搭載し、心拍や体温、歩数などユーザに密接に寄り添ったサービスを提供します。






講義ではこれらの端末上で動作するセンサーを利用したアプリの仕組みとプログラミングの学習を目標に、Phidget 社製 USB センサーを用いたアプリ作成の演習を行います。

http://www.phidgets.com/

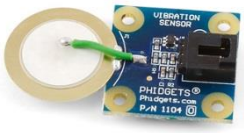
§2　Phidget 社製 USB センサー

Phidget 社製センサーには様々な種類があります。これらの全てのセンサーは USB 機器として提供されており、USB マウスと同じように手軽に扱うことができます。

以下は Phidget 社製センサーの種類の一例です。

■GPS

■振動



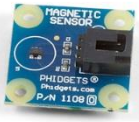
■圧力

■モーション

■明るさ

■距離

■磁気



■温度／湿度

■タッチ

■加速度／方位

■音

■気圧

講義では加速度／方位センサーを用いて演習を行います。

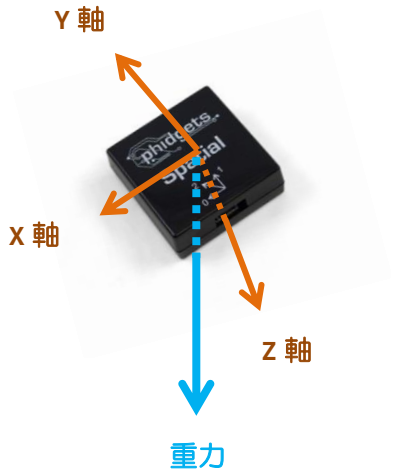
§3　加速度／方位センサー

講義では以下の加速度／方位センサーを用います。

■1042_0 - PhidgetSpatial 3/3/3 Basic

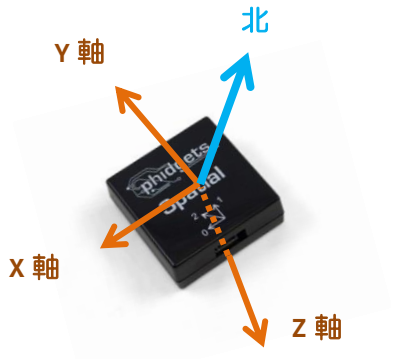
このセンサーは加速度（重力の方向）と方位（北の方角）をそれぞれ検出します。それぞれのデータは次の座標軸で表現されます。

■加速度データの表現



重力の方向（重力ベクトル）がセンサーの座標軸 3 軸で表現されます。ここで重力ベクトルの長さは 1 です。例えば、センサーを水平にしたとき検出されるデータは(0.0, 0.0, 1.0)です。左側を下にして立てたときは(1.0, 0.0, 0.0)です。

■方位データの表現



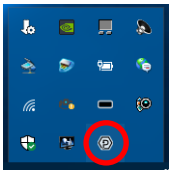
北の方角（方角ベクトル）がセンサーの座標軸 3 軸で表現されます。ここで方角ベクトルの長さは 1 です。例えば、センサーの Y 軸を北に向けたとき検出されるデータは(0.0, 1.0, 0.0)です。

§4　センサーの接続と動作確認

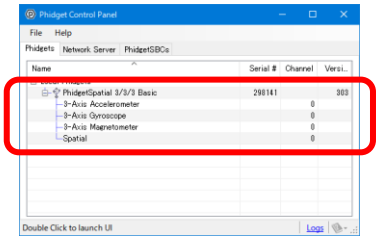
センサーを配布しますので前の方に集まってください。
以降、指示に従いながら動作確認を行いましょう。

1.センサーを USB へ接続しましょう。

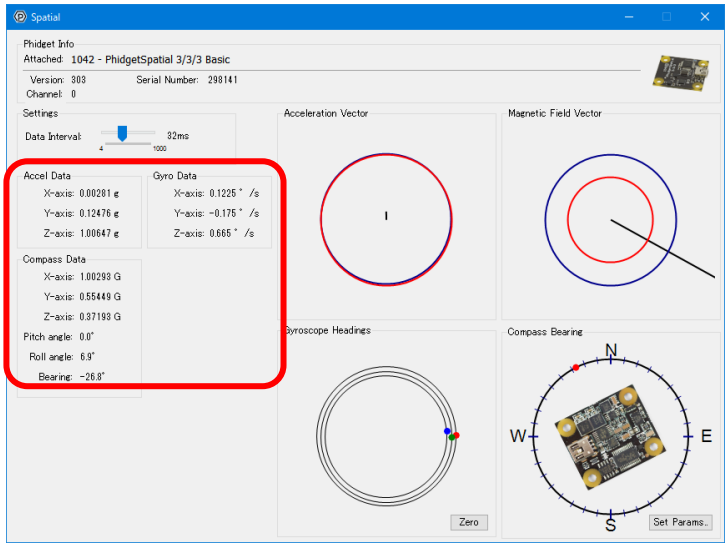
2.タスクバーから Phidget のアイコンを選択しましょう。



3.Phidget コントロールパネルからセンサーを選択して項目を表示しましょう。さらに、一番下の Spatial を選択しましょう。



4.選択したセンサーから取得されるデータが表示されます。



§5　センサーの接続と切断を検出してみましょう

センサーの各種イベントを取得してセンサーの接続や切断の検出、データの取得ができます。

ソースファイル名：Sample13_1.java

```
// ※HP よりインポート文をここへ貼り付けてください

// センサーの接続と切断
public class Sample13_1 extends Application
{
   private Spatial sp; // センサークラス型の変数

   // アプリケーションを開始するときに一度だけ実行されます
   public void start(Stage stage) throws Exception
   {
      // センサーを準備します
      sp = new Spatial();

      // センサーにイベントハンドラを登録します
      sp.addAttachListener(new myAttachListener());
      sp.addDetachListener(new myDetachListener());
      sp.addSpatialDataListener(new mySpatialDataListener());

      // センサーをオープンします
      sp.open();

      // シーンの生成／設定します
      Scene scene = new Scene(new FlowPane());

      // ステージを設定します
      stage.setScene(scene);
      stage.setTitle("センサー");

      // ステージを表示します
      stage.show();
   }

   // アプリケーションを終了するときに一度だけ実行されます
   public void stop() throws Exception
   {
      // センサーをクローズします
      sp.close();
   }

   // イベントハンドラ（センサーが接続されたとき）
   private class myAttachListener implements AttachListener
   {
      public void onAttach(AttachEvent e)
      {
```

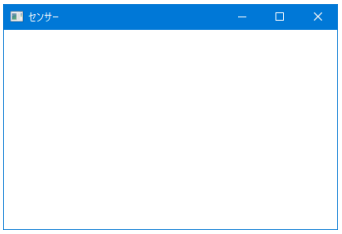
```
        System.out.println("●センサーが接続されました。");

        // データ取得間隔の設定
        try{
           sp.setDataInterval(32);
        }catch(PhidgetException pe){}
     }
  }

  // イベントハンドラ（センサーが切断されたとき）
  private class myDetachListener implements DetachListener
  {
     public void onDetach(DetachEvent e)
     {
        System.out.println("○センサーが切断されました。");
     }
  }

  // イベントハンドラ（センサーからデータを受け取ったとき）
  private class mySpatialDataListener implements SpatialSpatialDataListener
  {
     public void onSpatialData(SpatialSpatialDataEvent e)
     {
        System.out.println("※センサーからデータを受け取りました。");
     }
  }

  public static void main(String[] args)
  {
     launch(args);
  }
}
```



実行結果

```
●センサーが接続されました。          ← センサーを接続します
※センサーからデータを受け取りました。
※センサーからデータを受け取りました。
            :
※センサーからデータを受け取りました。
○センサーが切断されました。          ← センサーを切断します
```

■センサーを表現するクラス Spatial

センサーはクラス Spatial により表現されます。このクラスにはイベントハンドラの設定など各種設定を行うメソッドが準備されています。

・センサーの生成　　→ new Spatial();
・接続イベントハンドラの設定　　→ addAttachListener(…);
・切断イベントハンドラの設定　　→ addDetachListener(…);
・データイベントハンドラの設定　　→ addSpatialDataListener(…);
・センサーのオープン　　→ open();
・センサーのクローズ　　→ close();

センサーを生成した後、各イベントハンドラの設定を行います。センサーのオープンを行うとイベントハンドラの処理が開始され、センサーのクローズを行うまで継続されます。

■接続イベント（AttachEvent）とは

センサーが USB に接続されたときに発生するイベントです。接続イベントはクラス AttachEvent により表現され、接続されたセンサー情報などイベント関連の情報を保持します。

■切断イベント（DetachEvent）とは

センサーが USB から切断されたときに発生するイベントです。切断イベントはクラス DetachEvent により表現され、切断されたセンサー情報などイベント関連の情報を保持します。

■データイベント（SpatialSpatialDataEvent）とは

センサーからデータが来たときに発生するイベントです。データイベントはクラス SpatialSpatial-DataEvent により表現され、**重力データ**や**方位データ**などのイベント関連の情報を保持します。

■接続イベントを処理するイベントハンドラインタフェース AttachListener

接続イベントはイベントハンドラクラスで受け取り、対応する処理を行います。

1．AttachListener インタフェースを実装してイベントハンドラクラスを宣言
2．継承される void onAttach(AttachEvent e);メソッドをオーバーライドして処理を記述

※発生したイベントがメソッドの引数 e に渡されて呼び出されます

〔コード例〕

```
1. class myAttachListener implements AttachListener{
2.   public void onAttach(AttachEvent a)
3.   {
4.     // ここにイベントに対応する処理を記述します
5.   }
6. }
```

■切断イベントを処理するイベントハンドラインタフェース DetachListener

切断イベントはイベントハンドラクラスで受け取り、対応する処理を行います。

1．DetachListener インタフェースを実装してイベントハンドラクラスを宣言

2．継承される void onDetach(DetachEvent e);メソッドをオーバーライドして処理を記述

■データイベントを処理するイベントハンドラインタフェース SpatialSpatialDataListener

データイベントはイベントハンドラクラスで受け取り、対応する処理を行います。

1．SpatialSpatialDataListener インタフェースを実装してイベントハンドラクラスを宣言

2．継承される void onSpatialData(SpatialSpatialDataEvent e);メソッドをオーバーライドして処理を記述

■start()メソッドと stop()メソッド

ウィンドウアプリケーションは main()メソッドから処理が始まります。ここで Application クラスから継承された launch()メソッドが実行されます。launch()メソッドはアプリケーション開始の準備を行いながら内部で start()メソッドを実行し、アプリケーションを開始します。

〔開始〕→ main() → launch() → start() →〔実行中〕→ stop() →〔終了〕

その後、クローズボタンなどによりアプリケーションが終了するときには stop()メソッドが実行され、アプリケーションは終了します。

※start()メソッドや stop()メソッドはともに Application クラスから継承されたメソッドであり、オーバーライドして各自の処理を記述します。

■利用したクラス／インタフェースの一覧

Spatial クラス	
Spatial(){…}	センサーを生成します。
void addAttachListener(AttachListener l){…}	接続イベントハンドラを設定します。
void addDetachListener(DetachListener l){…}	切断イベントハンドラを設定します。
void addSpatialDataListener(SpatialSpatialDataListener l){…}	データイベントハンドラを設定します。
void open(){…}	センサーをオープンし、イベント処理を開始します。
void close(){…}	センサーをクローズし、イベント処理を終了します。
void setDataInterval(int i){…}	データの取得間隔を設定します。
AttachListener インタフェース	
void onAttach(AttachEvent e);	接続イベントが発生したときに実行されます。
DetachListener インタフェース	
void onDetach(DetachEvent e);	切断イベントが発生したときに実行されます。
SpatialSpatialDataListener インタフェース	
void onSpatialData(SpatialSpatialDataEvent e);	データが発生したときに実行されます。

§6　重力（傾き）データを取得してみましょう

センサーの重力データを取得して、センサーの傾きを調べることができます。

ソースファイル名：Sample13_2.java

```
// ※HP よりインポート文をここへ貼り付けてください

// センサーから傾きデータを取得
public class Sample13_2 extends Application
{
   private Spatial sp; // センサークラス型の変数

   // アプリケーションを開始するときに一度だけ実行されます
   public void start(Stage stage) throws Exception
   {
      // センサーを準備します
      sp = new Spatial();

      // センサーにイベントハンドラを登録します
      sp.addAttachListener(new myAttachListener());
      sp.addDetachListener(new myDetachListener());
      sp.addSpatialDataListener(new mySpatialDataListener());

      // センサーをオープンします
      sp.open();

      // シーンの生成／設定します
      Scene scene = new Scene(new FlowPane());

      // ステージを設定します
      stage.setScene(scene);
      stage.setTitle("センサー");

      // ステージを表示します
      stage.show();
   }

   // アプリケーションを終了するときに一度だけ実行されます
   public void stop() throws Exception
   {
      // センサーをクローズします
      sp.close();
   }

   // イベントハンドラ（センサーが接続されたとき）
   private class myAttachListener implements AttachListener
   {
      public void onAttach(AttachEvent e)
      {
         System.out.println("●センサーが接続されました。");
```

```
        // データ取得間隔の設定
        try{
           sp.setDataInterval(32);
        }catch(PhidgetException pe){}
     }
  }

  // イベントハンドラ（センサーが切断されたとき）
  private class myDetachListener implements DetachListener
  {
     public void onDetach(DetachEvent e)
     {
        System.out.println("〇センサーが切断されました。");
     }
  }

  // イベントハンドラ（センサーからデータを受け取ったとき）
  private class mySpatialDataListener implements SpatialSpatialDataListener
  {
     public void onSpatialData(SpatialSpatialDataEvent e)
     {
        // 傾きデータ（重力方向）を取り出します
        double[] data = e.getAcceleration();
        System.out.println("x="+data[0]+" y="+data[1]+" z="+data[2]);
     }
  }

  public static void main(String[] args)
  {
     launch(args);
  }
}
```

実行結果

```
●センサーが接続されました。          ← センサーを接続します
x=0.55078 y=-0.22607 z=0.81543
x=0.54199 y=-0.30908 z=0.81494
            :
x=0.50732 y=-0.26367 z=0.83447
〇センサーが切断されました。          ← センサーを切断します
```

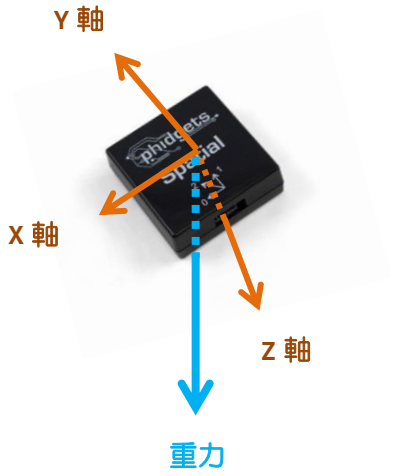
■重力ベクトルを取得するには?

センサーからのデータイベントを表すクラス SpatialSpatialDataEvent に保持されます。データイベントハンドラが呼ばれたとき、その引数にこのクラスのオブジェクト e が渡されます。次のようにして重力ベクトルを取得します。

```
double[] data = e.getAcceleration();
```

getAcceleration()メソッドは、重力ベクトルを double 型の配列として返します。

data[0]　→ 重力ベクトルの x 軸座標値
data[1]　→ 重力ベクトルの y 軸座標値
data[2]　→ 重力ベクトルの z 軸座標値



■利用したクラス／インタフェースの一覧

SpatialSpatialDataEvent クラス

double[] getAcceleration(){…}	重力ベクトルを取得します。

§7　方位データを取得してみましょう

センサーの方位データを取得して、北の方角を調べることができます。

ソースファイル名：Sample13_3.java

```
// ※HP よりインポート文をここへ貼り付けてください

// センサーから方位データを取得
public class Sample13_3 extends Application
{
   private Spatial sp; // センサークラス型の変数

   // アプリケーションを開始するときに一度だけ実行されます
   public void start(Stage stage) throws Exception
   {
      // センサーを準備します
      sp = new Spatial();

      // センサーにイベントハンドラを登録します
      sp.addAttachListener(new myAttachListener());
      sp.addDetachListener(new myDetachListener());
      sp.addSpatialDataListener(new mySpatialDataListener());

      // センサーをオープンします
      sp.open();

      // シーンの生成／設定します
      Scene scene = new Scene(new FlowPane());

      // ステージを設定します
      stage.setScene(scene);
      stage.setTitle("センサー");

      // ステージを表示します
      stage.show();
   }

   // アプリケーションを終了するときに一度だけ実行されます
   public void stop() throws Exception
   {
      // センサーをクローズします
      sp.close();
   }

   // イベントハンドラ（センサーが接続されたとき）
   private class myAttachListener implements AttachListener
   {
      public void onAttach(AttachEvent e)
      {
         System.out.println("●センサーが接続されました。");
```

```
        // データ取得間隔の設定
        try{
           sp.setDataInterval(32);
        }catch(PhidgetException pe){}
     }
  }

  // イベントハンドラ（センサーが切断されたとき）
  private class myDetachListener implements DetachListener
  {
     public void onDetach(DetachEvent e)
     {
        System.out.println("〇センサーが切断されました。");
     }
  }

  // イベントハンドラ（センサーからデータを受け取ったとき）
  private class mySpatialDataListener implements SpatialSpatialDataListener
  {
     public void onSpatialData(SpatialSpatialDataEvent e)
     {
        // 方位データを取り出します
        double[] data = e.getMagneticField();
        System.out.println("x="+data[0]+" y="+data[1]+" z="+data[2]);
     }
  }

  public static void main(String[] args)
  {
     launch(args);
  }
}
```

実行結果

```
●センサーが接続されました。          ← センサーを接続します
x=0.60903 y=-0.26513 z=-0.02348
x=0.60903 y=-0.26588 z=-0.02273
            :
x=0.60752 y=-0.26513 z=-0.02273
〇センサーが切断されました。          ← センサーを切断します
```

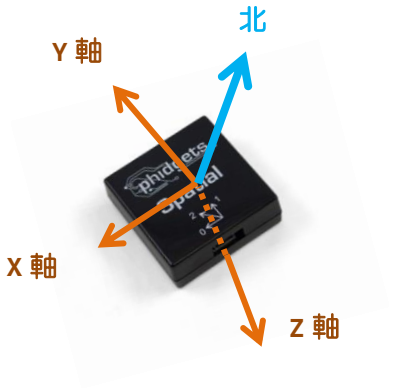
■方位ベクトルを取得するには?

センサーからのデータイベントを表すクラス SpatialSpatialDataEvent に保持されます。データイベントハンドラが呼ばれたとき、その引数にこのクラスのオブジェクト e が渡されます。次のようにして重力ベクトルを取得します。

```
double[] data = e.getMagneticField();
```

getMagneticField()メソッドは、方位ベクトルを double 型の配列として返します。

data[0] → 方位ベクトルの x 軸座標値
data[1] → 方位ベクトルの y 軸座標値
data[2] → 方位ベクトルの z 軸座標値



■利用したクラス／インタフェースの一覧

SpatialSpatialDataEvent クラス

double[] getMagneticField(){…}	データセットから方位ベクトルを取得します。