모바일 시스템 프로그래밍

09 Mobile Sensing Pipeline 2

2017 1학기

강승우

Computer Science and Engineering, KOREATECH

Accelerometer를 이용한 활동량 모니터링 (Step Monitor version 2)

Step Monitor version 1의 문제

- 센서 데이터 중 y축 값만 보는 것으로 충분한가?
  - 문제점: 실제 걷는 중이더라도 y축 방향으로 움직임이 거의 없다면, 가속도의 변화가 크지 않을 것이고 그러면 걸음 수가 증가하지 않을 것이다
- 현재 값과 바로 이전 값의 차이를 보는 것으로 step을 구분하는 것이 적당한가?
  - 문제점: 순간적인 움직임에도 걸음수가 2-3 씩 증가하는 경우가 생긴다
    - SENSOR_DELAY_NORMAL인 경우 Nexus 5 기준 약 0.2초 간격으로 데이터 업데이트
- Threshold를 어떻게 정하는가?
  - 문제점: 실제 걸음 중의 가속도 변화 정도가 threshold를 정하는데 반영이 안 되어 있다

Step Monitor version 1의 개선 방향

- 센서 데이터 중 y축 값만 보는 것으로 충분한가?
  → 3축 어느 방향의 움직임이든 이에 의해 가해지는 가속도를 고려할 필요가 있음

- 현재 값과 바로 이전 값의 차이를 보는 것으로 step을 구분하는 것이 적당한가?
  → 일정 시간의 데이터를 모아서 볼 필요가 있음

- Threshold를 어떻게 정하는가?
  → 실제 걷는 중에 가속도 값의 변화 추이를 살펴볼 필요가 있음

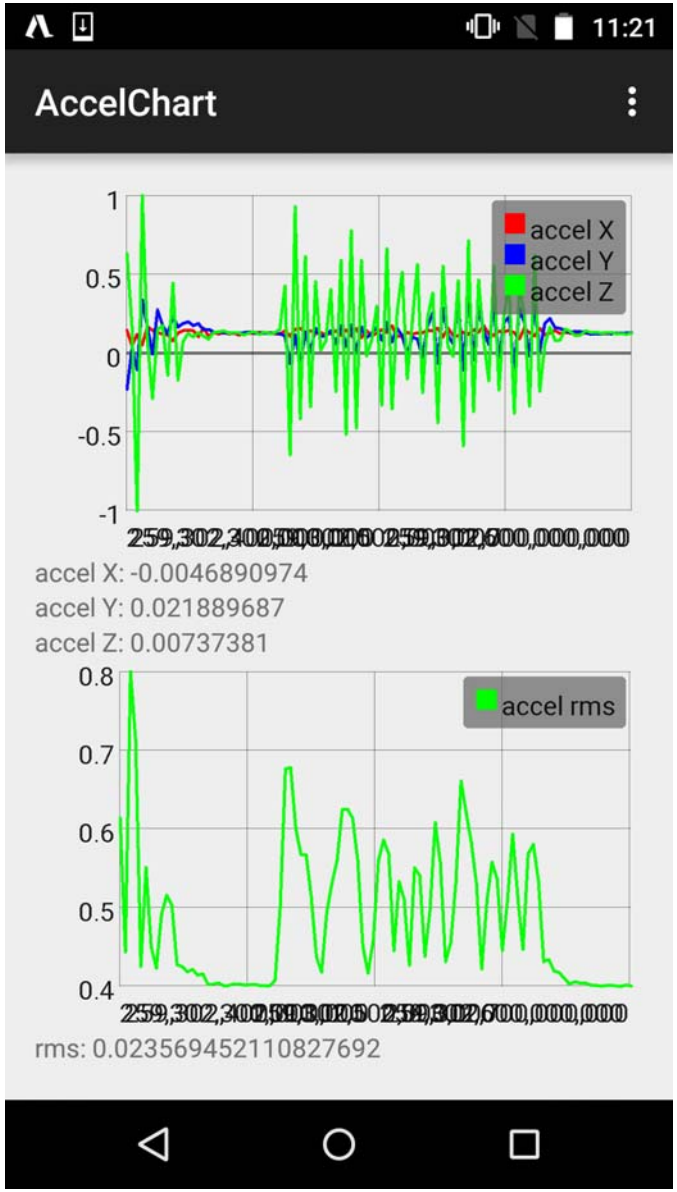
Step Monitor version 1 개선하기

- Step Monitor version 1을 개선하기 위해 가속도 데이터를 관찰해보자

- 데이터 변화 추이를 관찰하기 위해 그래프로 시각화
  - 예제 프로젝트 이름: MSP12AccelChart

  - GraphView 라이브러리 이용
    - Open source graph plotting library for Android
    - http://www.android-graphview.org/

    ➢ 이외에도 사용 가능한 다른 라이브러리도 있음

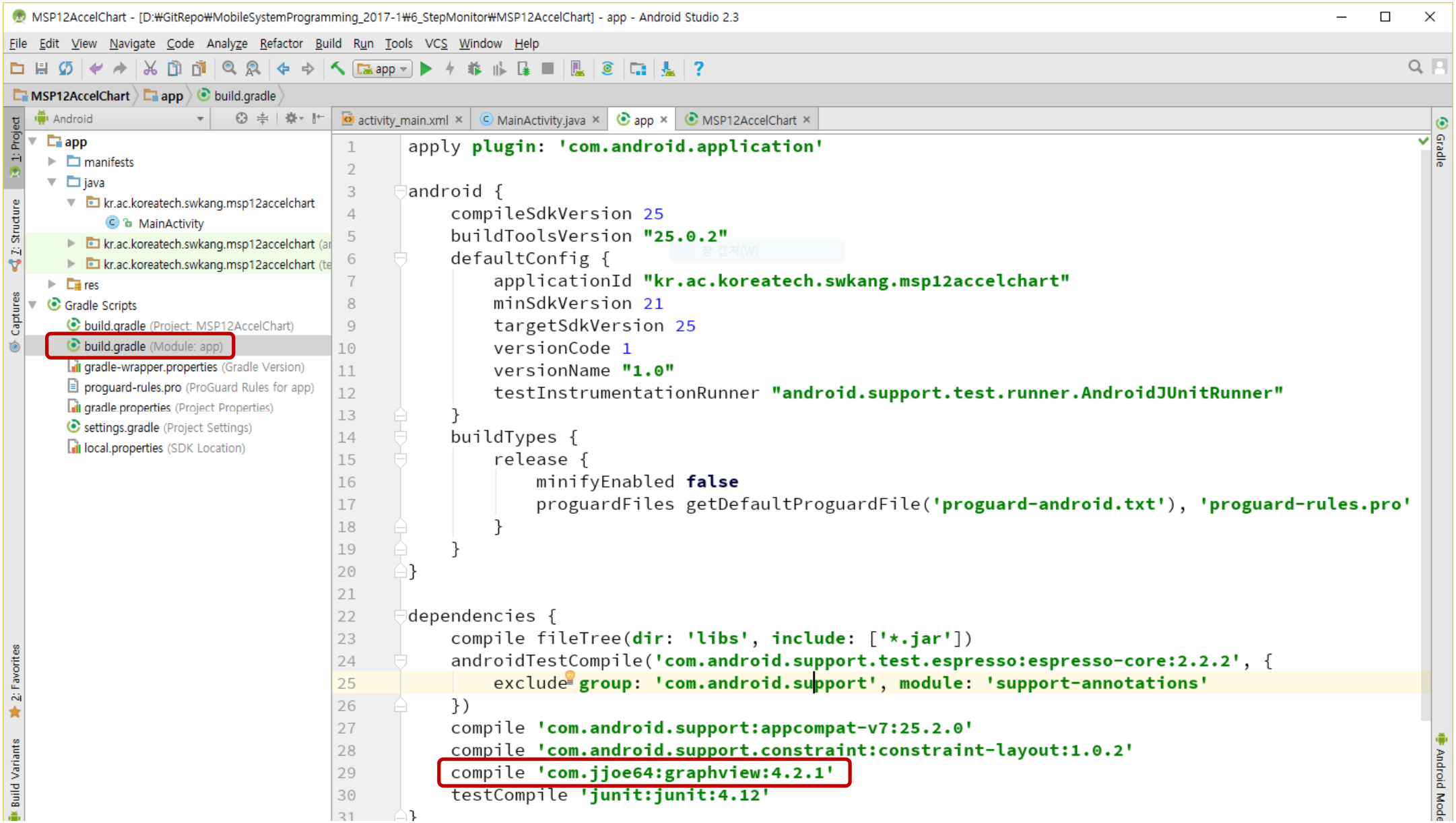


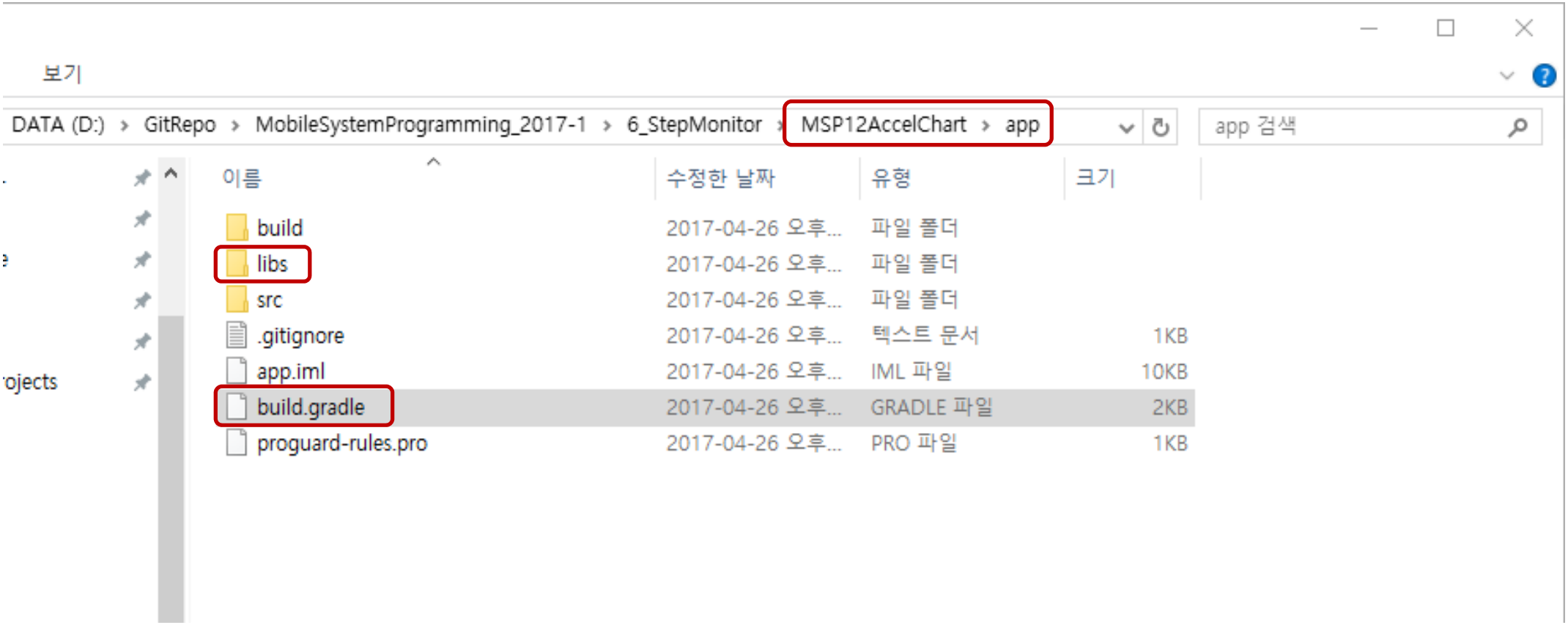
GraphView 라이브러리 이용

- 2가지 방법
  - Gradle dependency 추가
    - 안드로이드 스튜디오 프로젝트 폴더 아래 app 폴더 안에 있는 build.gradle 파일의 dependencies 블록에 아래 내용 추가

      compile 'com.jjoe64:graphview:4.2.1'

  - 라이브러리 jar 파일을 직접 추가
    - 안드로이드 스튜디오 프로젝트 폴더의 libs 폴더에 복사


MSP12AccelChart - [D:₩GitRepo₩MobileSystemProgramming_2017-1₩6_StepMonitor₩MSP12AccelChart] - app - Android Studio 2.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
MSP12AccelChart app build.gradle
Android
app
manifests
java
kr.ac.koreatech.swkang.msp12accelchart
MainActivity
res
Gradle Scripts
build.gradle (Project: MSP12AccelChart)
build.gradle (Module: app)
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)
activity_main.xml
MainActivity.java
app
MSP12AccelChart
apply plugin: 'com.android.application'
android {
compileSdkVersion 25
buildToolsVersion "25.0.2"
defaultConfig {
applicationId "kr.ac.koreatech.swkang.msp12accelchart"
minSdkVersion 21
targetSdkVersion 25
versionCode 1
versionName "1.0"
testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
}
buildTypes {
release {
minifyEnabled false
proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
}
}
}
dependencies {
compile fileTree(dir: 'libs', include: ['*.jar'])
androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
exclude group: 'com.android.support', module: 'support-annotations'
})
compile 'com.android.support:appcompat-v7:25.2.0'
compile 'com.android.support.constraint:constraint-layout:1.0.2'
compile 'com.jjoe64:graphview:4.2.1'
testCompile 'junit:junit:4.12'
}
Gradle
Android Mode
1: Project
7: Structure
Captures
2: Favorites
Build Variants

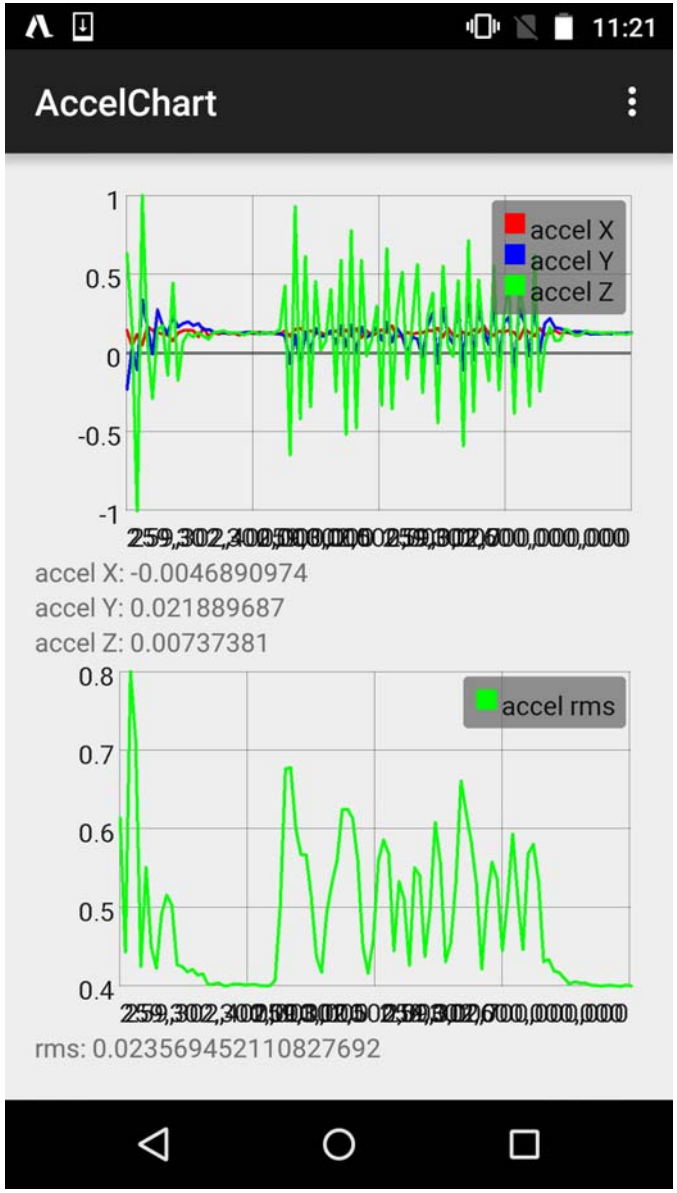


- **참고**
  - Android Studio에서 외부 안드로이드 라이브러리 이용하기
    http://sunghwanjo94.blogspot.kr/2015/07/jcenter.html

가속도 데이터 관찰

- 중력 가속도를 제외하고 폰에 가해지는 힘에 의한 가속도만 관찰해보자
  → Sensor.TYPE_LINEAR_ACCELERATION 이용

- 상단 그래프
  - 각 축 별로 linear acceleration 값을 선 그래프로 표시
- 하단 그래프
  - 폰에 가해지는 가속도의 크기를 보기 위해 3축 가속도 값의 RMS 값을 선 그래프로 표시
  - RMS = sqrt(x*x + y*y + z*z)
    - Root Mean Square



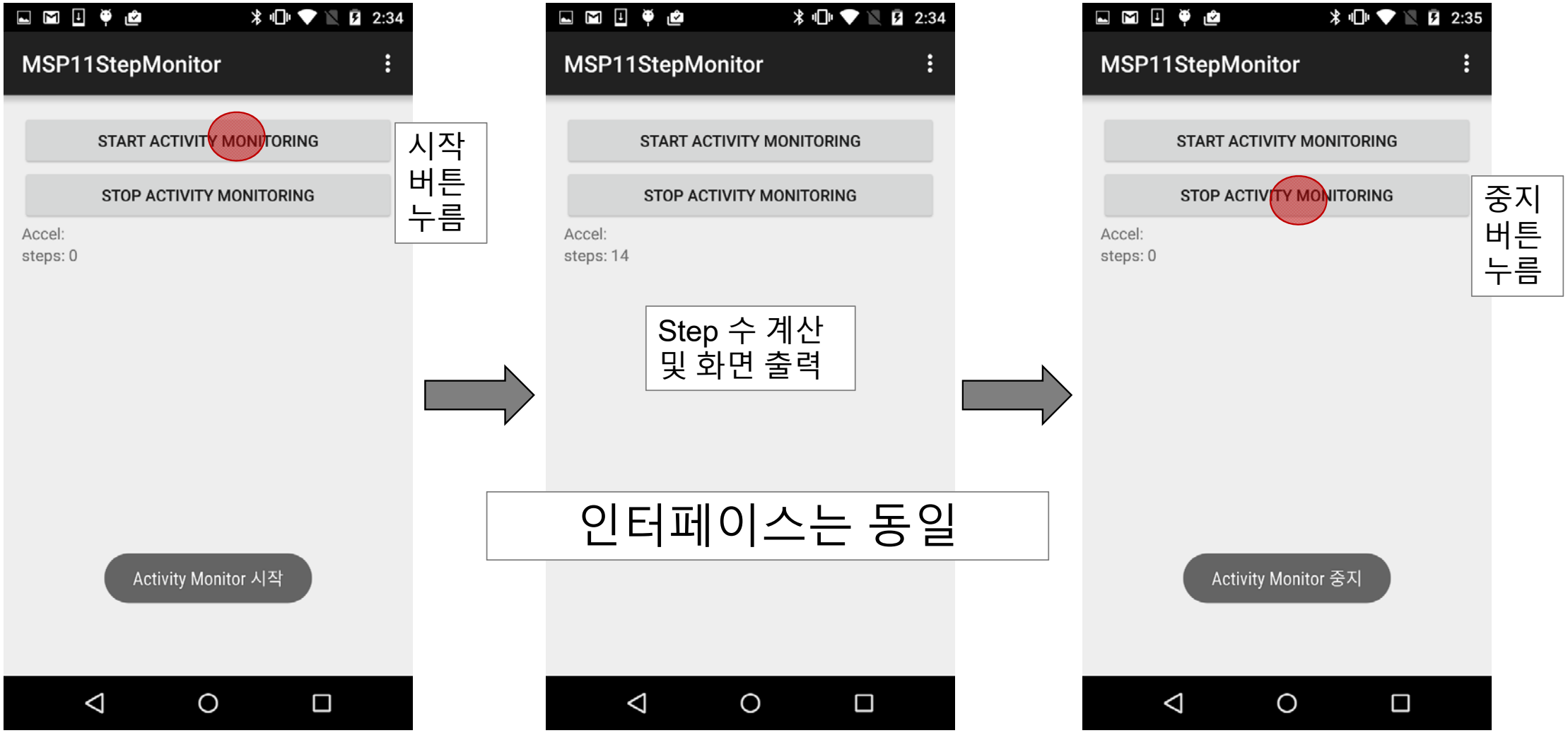
Step Monitor version 1 개선 방향

- Y축 가속도만 고려하는 것이 아니라 폰에 가해지는 전체 가속도를 고려
  - Linear acceleration 값의 RMS 값을 계산하여 이용

- 일정 시간의 데이터를 모아서 이용
  - 1초 동안의 가속도 RMS의 평균을 계산하여 이용

- RMS 값의 변화 추이를 관찰하여 Threshold 결정
  - 걷는 동안 RMS 값과 움직이지 않는 동안 RMS 값의 차이 관찰

Step Monitor ver. 2 (예제 프로젝트 이름: MSP13StepMonitor2)



Step Monitor ver. 2 데이터 처리 과정

- Sensor data collection
  - 3축 가속도 데이터 수집
- Feature extraction
  - 가속도 데이터 업데이트 마다 RMS 계산
  - 1초간 데이터 buffering
  - 1초 동안의 RMS 값이 모였을 때, 평균 RMS 계산
- Classification
  - Step 유무 판단
    - 평균 RMS가 일정 threshold보다 크면 step이 있었다고 판단
    - 작으면 없었다고 판단
  - Step이 있었다고 판단되면 step count 증가
    - 가정: 초당 걸음수가 일정하다고 가정하여 그 수만큼 증가

Code

- 변경 코드 (StepMonitor.java)
  - onSensorChanged()
  - computeSteps()
  - 위 두 함수만 변경

  - 소스 코드를 직접 보자!

```
// 센서 데이터가 업데이트 되면 호출
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_LINEAR_ACCELERATION) {

        //*** 데이터 업데이트 주기 확인 용 코드 ***//
        // SENSOR_DELAY_NORMAL, SENSOR_DELAY_UI,
        // SENSOR_DELAY_GAME, SENSOR_DELAY_FASTEST로
        // 변경해가면서 로그(logcat)를 확인해 볼 것
        currT = event.timestamp;
        double dt = (currT - prevT)/1000000;
        // logcat에 로그를 출력하려면 아래 code line의 주석을 해제
            //Log.d(LOGTAG, "time difference=" + dt);
        prevT = currT;
        //*************************************

        //***** sensor data collection *****//
        // event.values 배열의 사본을 만들어서 values 배열에 저장
        float[] values = event.values.clone();

        // simple step calculation
        computeSteps(values);
    }
}
```

```
// a simple inference for step count
private void computeSteps(float[] values) {
    double avgRms = 0;

    //***** feature extraction *****//
    // calculate feature data:
    // 여기서는 3축 가속도 데이터의 RMS 값의 1초 간의 평균값을 이용

    // 1. 현재 업데이트 된 accelerometer x, y, z 축 값의 Root Mean Square 값 계산
    double rms = Math.sqrt(values[0] * values[0] + values[1] * values[1] + values[2] * values[2]);
```

```
// 2. 위에서 계산한 RMS 값을 rms 값을 저장해 놓는 배열에 넣음
// 배열 크기는 1초에 발생하는 가속도 데이터 개수 (여기서는 5)
if(rmsCount < NUMBER_OF_SAMPLES) {
    rmsArray[rmsCount] = rms;
    rmsCount++;
} else if(rmsCount == NUMBER_OF_SAMPLES) {
      // 3. 1초간 rms 값이 모였으면 평균 rms 값을 계산
      double sum = 0;
      // 3-1. rms 값들의 합을 구함
      for(int i = 0; i < NUMBER_OF_SAMPLES; i++) {
       sum += rmsArray[i];
    }
    // 3-2. 평균 rms 계산
    avgRms = sum / NUMBER_OF_SAMPLES;
    Log.d(LOGTAG, "1sec avg rms: " + avgRms);

    // 4. rmsCount, rmsArray 초기화: 다시 1초간 rms sample을 모으기 위해
    rmsCount = 0;
    for(int i = 0; i < NUMBER_OF_SAMPLES; i++) {
        rmsArray[i] = 0;
    }

    // 5. 이번 업데이트로 계산된 rms를 배열 첫번째 원소로 저장하고 카운트 1증가
    rmsArray[0] = rms;
    rmsCount++;
}
```

```
        //***** classification *****//
        // check if there is a step or not:
        // 1. 3축 가속도 데이터의 1초 평균 RMS 값이 기준 문턱값을 넘으면 step이 있었다고 판단함
        if(avgRms > AVG_RMS_THRESHOLD) {
            // 1-1. step 수는 1초 걸음 시 step 수가 일정하다고 가정하고, 그 값을 더해 줌
            steps += NUMBER_OF_STEPS_PER_SEC;
            Log.d(LOGTAG, "steps: " + steps);

            // if step counts increase, send steps data to MainActivity
            Intent intent = new Intent("kr.ac.koreatech.msp.stepmonitor");
            // 걸음수는 정수로 표시되는 것이 적합하므로 int로 형변환
            intent.putExtra("steps", (int)steps);
            // broadcast 전송
            sendBroadcast(intent);
        }
    }
```

생각해볼 문제

- 왜 1초의 시간인가?
  - 걷는 중이라는 것을 1초간 가속도 RMS 평균 값으로 판단하고 있음
  - 더 짧게 하거나 더 길게 하는 것이 더 좋을까?
- 왜 RMS의 평균값인가?
  - 더 보면 좋은 다른 것이 있을까? 아니면 다른 것이 더 나을까?
- 가속도 RMS의 Threshold 기반으로 걸음 여부 판단
  - 폰을 들어 올리거나 폰을 쥐고 움직이는 경우도 걸음으로 오인식 할 수 있음
- 시간당 걸음 수 가정
  - 현재는 항상 분당 90보를 걷는다고 가정
  - 천천히 걷거나 빨리 걷는 경우 오차 발생

GPS, WiFi를 이용한 위치 모니터링 (Location Tracker)

Location Tracker

- 이동한 장소 정보(이름), 이동 시각, 체류 시간 등을 기록하는 애플리케이션
- 예
  - LifeMap
    - 연세대학교 Mobed Research Group에서 만든 앱
  - Moves
    - 활동량 및 위치 트래킹 앱

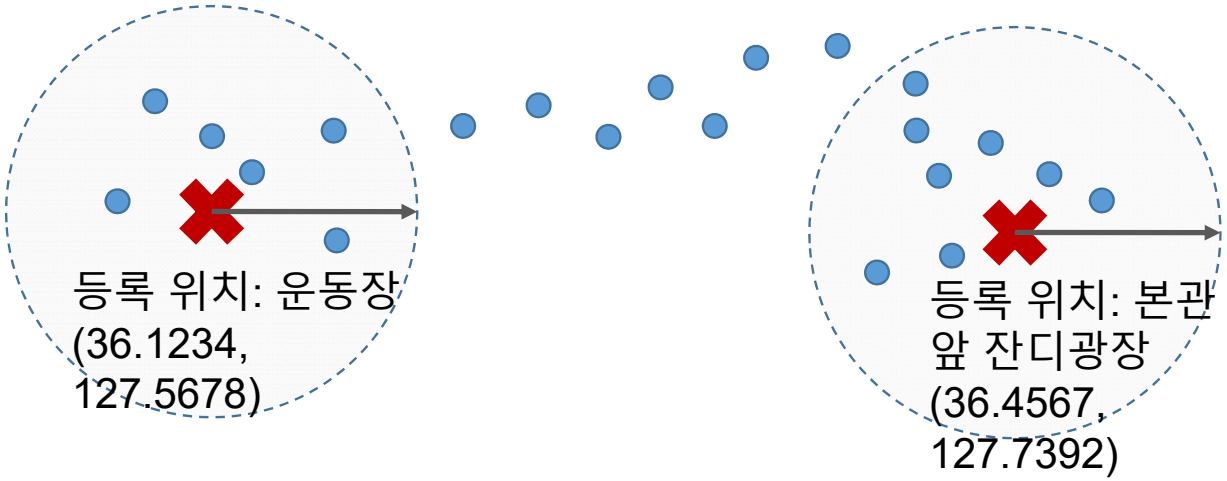
LifeMap
https://play.google.com/store/apps/details?id=com.mobed.lifemap

Our version of Location Tracker

- Simple heuristic for our Location Tracker
  - 5분 이상 동일 장소에 머무르는 경우 이를 기록하는 tracker
  - 실내/실외 위치
    - 실외 3곳, 실내 3곳의 위치 정보를 미리 등록
      - 실외: GPS 위도, 경도 기반
      - 실내: WiFi AP ID, RSSI 기반
  - 현재 위치를 계속 모니터링 하면서 등록된 장소에 5분 이상 있는지 검사
    - 5분 이상 있으면 머무르기 시작한 시점 기록
    - 다른 곳으로 이동한 경우 해당 장소에서 머무른 시간 계산하여 기록
    - 예)
    - 2017.04.27 13:30, 운동장 (30분)
    - 2017.04.27 16:50, A309 (50분)

등록된 장소 여부 판단 기준 (실외 경우)

- GPS 위도, 경도 좌표 사이의 거리
  - 등록된 위치의 좌표에서 특정 반경 이내의 좌표는 해당 위치에 있는 것으로 판단
  - 반경은 장소에 따라 크게 잡을 필요가 있는 곳도 있고, 작게 잡아도 될 곳도 있을 것임
    - GPS 위치의 오차 범위도 고려할 필요가 있음



두 위치 좌표 사이의 거리 계산

- Location 클래스의 distanceBetween() 메소드 혹은 distanceTo() 메소드 이용 가능

https://developer.android.com/reference/android/location/Location.html?hl=ko

등록된 장소 여부 판단 기준 (실내 경우)

- AP 정보 유사도
  - 현재 스캔한 AP 정보가 사전 등록된 위치의 AP 정보와 유사한 경우 등록된 위치에 있는 것으로 판단
  - 유사함의 기준 (MSP06IndoorProximityAlert 예제를 조금 개선한 방식)
    - RSSI top-k AP 중 등록된 AP가 n개 이상 있고, 그 RSSI 값이 일정 값 이내인 경우
      - 위치 등록 시 RSSI top-3 AP를 등록
      - 현재 스캔한 AP 중 RSSI top-3 AP가 등록된 3개 AP와 2개 이상이 일치하고, 그 RSSI 값의 차이가 20dBm 이내인 경우

등록 위치: A309

AP1	-30dBm
AP2	-55dBm
AP3	-60dBm

현재 스캔 결과

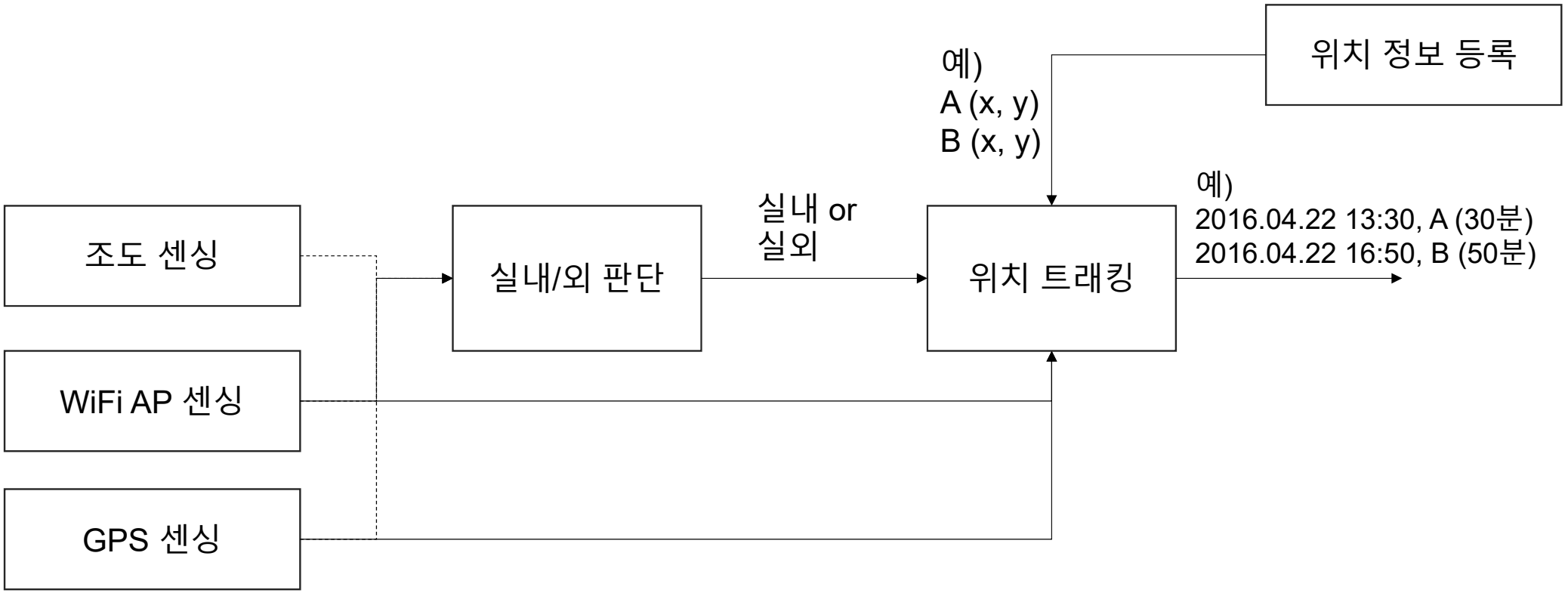
AP1	-39dBm
AP3	-50dBm
AP4	-60dBm
AP2	-65dBm
AP5	-42dBm

⇨ 현재 위치는 A309로 판단

실내/외 여부 판단 기준

- 현재 위치의 GPS 데이터의 정확도
  - 가정: 실내에서는 GPS 위치 정확도가 낮을 것이다
  - Location 클래스 getAccuracy() 메소드
    - Get the estimated accuracy of this location, in meters.
    - We define accuracy as the radius of 68% confidence. In other words, if you draw a circle centered at this location's latitude and longitude, and with a radius equal to the accuracy, then there is a 68% probability that the true location is inside the circle.
- 현재 위치에서 스캔된 AP 개수와 RSSI
  - 가정: 실외에서는 스캔되는 AP가 별로 없고 RSSI도 작을 것이다
- 조도
  - 가정: 낮 동안은 실외가 실내보다 조도가 높을 것이다
    - 실외에 있지만 폰을 주머니나 가방에 둔 경우는?

Example Sensing Pipeline for Location Tracker



생각해 볼 문제

- 1. location tracking 시 센싱 주기는 어떻게 정할까?
  - 얼마 간격으로 센싱을 해야 할까?
  - 5분 동안 한 장소에 머무른다는 것을 감지할 수 있는 정도의 센싱 주기
- 2. 센싱을 계속 할 필요가 있을까?
  - 실내/외 검사 후 실외라고 판단되는 경우, location tracking은 GPS 데이터로 하므로, 조도 센싱, WiFi AP 센싱을 계속할 필요가 없음
    - 실내라고 판단되는 경우도 마찬가지
  - 그런데, GPS로 tracking 중 실내로 들어갈 수도 있을 텐데, 조도 센싱, WiFi AP 센싱을 멈춰버리면 이런 변화를 어떻게 감지하지?
- 3. 실내/외 판단의 구체적인 방법은 어떻게 정할까?
  - GPS 정확도가 얼마이면?
  - WiFi AP 개수가 얼마이면? 신호강도는? 실외에서는 정말 AP가 안 보이나?
  - 조도가 얼마이면?
  - 이 3가지 정보를 어떻게 조합하여 판단?
- 4. 등록되지 않은 장소에 대해서도 기록을 한다면?

과제

- Step Monitor ver. 2 개선
  - 18번 슬라이드의 생각해 볼 문제 중
    폰을 들어 올리는 등의 짧은 움직임이 걸음으로 오인식 되는 문제를 해결하기 위한 방법을 만드시오
    - Flow chart 형태로 알고리즘을 디자인, 왜 그렇게 디자인 했는지 이유/근거 설명
- Location Tracker 디자인
  - 26번 슬라이드의 생각해 볼 문제 중
    1, 2번 문제에 대한 해결 방안을 만드시오
    - 1번은 주기를 얼마로 설정할 것인지, 그렇게 생각한 이유는 무엇인지 기술
    - 2번은 flow chart 형태로 알고리즘을 디자인, 왜 그렇게 디자인 했는지 이유/근거 설명

- 위 내용에 대한 보고서를 작성하여 제출 (기한: 5/11)
  - 기본적으로 2인 1조로 하되 사정상 1인으로 해야 한다면 그렇게 해도 무방