PDIoT Coursework Notes

# Introduction

The PDIoT coursework involves using an inertial sensor to detect steps when walking. You will use an embedded development board to interface with an inertial sensor. The dev board should then communicate wirelessly with an Android app, which you will develop.

The rest of this document is split into sections corresponding to each part of the practical and provides setup instructions and suggested first steps.

# Git repository

This document are supporting files are available from the following GitHub repository: <https://github.com/specknet/pdiot-practical>.

You are strongly encouraged to use version control for your own work.

# Data Analysis

Initially, we recommend developing step tracking algorithms using existing walking data and running them offline on a PC. Once you are happy with your algorithm it can be ported to your Android app to perform steptracking on live sensor data.

Python and Jupyter Notebook provides a rapid way to explore sensor data using various data analysis techniques. Further information for this section is contained in the accompanying Jupyter notebook.

## Requirements

Python 2.7 installation with Jupyter notebook, Pandas and Matplotlib. Anaconda python distribution is recommended.

<https://www.anaconda.com/distribution/>

# Android

The practical will require you to develop an Android app, which will interface to your development board using Bluetooth LE and provide a user interface showing the step count. You will also be provided with a basic app which can be used to record sensor data for offline analysis.

## Development Environment

You should use Android Studio.

<https://developer.android.com/studio/>

## Phone

We use Xiomi Redmi 4A or 5A phones and can lend one if required. Other phones may work for the practical but there can be Bluetooth reliability issues with other devices.

## Data Collection App

An app is provided to record accelerometer and gyroscope data to a CSV file. Please use this to collect walking data to ensure that all groups use a common file format and include appropriate metadata.

Initially you can continue to use the Nordic cube sensor and switch over to your own mbed implementation during the second half of the practical.

## BLE Introduction

Bluetooth Low Energy (BLE) provides a cheap and reliable way for low power devices to communicate. Devices advertise one of more services, which themselves contain a number of characteristics. For example, a heart rate monitor may provide a service which contains a characteristic which will send the current pulse rate. Characteristics can either be readable, writable or allow notifications, which means that new data will be streamed over BLE when it is available. This is the mode that we use to send accelerometer and gyroscope data from the Cube.

The Nordic Semiconductor NRF Connect app (available on play/app store) will allow you to connect to BLE devices and interrogate the services and characteristics that they provide. It can also send/receive data and log communications to a file, which can be useful for debugging. Try this with the sensor cube to see how the sensor data is sent over BLE. Gyroscope, accelerometer and magnetometer data are packed into an 18-byte packet, where each axis of each sensor requires 2 bytes to send a 16-bit value.

Details on the architecture of the system, the characteristics for different services and the packet structure can be found here: <https://nordicsemiconductor.github.io/Nordic-Thingy52-FW/documentation/firmware_architecture.html>

## BLE on Android

This repository contains the PDIoT data collection app, which you can use as an example of BLE communication on Android. We use the RXAndroidBLE library which simplifies much of the communication code. Note that this requires Java 8 support (enabled in build.gradle as shown below).

<https://polidea.github.io/RxAndroidBle>

build gradle:

**android {**

**compileOptions {**

**sourceCompatibility JavaVersion.VERSION\_1\_8**

**targetCompatibility JavaVersion.VERSION\_1\_8**

**}**

**}**

## Permissions

To make the data collection app work correctly, you’ll need to enable *location* and *storage* permissions in *settings/apps/permissions*. Location permissions is required when scanning for BLE devices. If you don’t have this you’ll see a *BLE Scanning Error* message when starting the app.

## Goal

You will need to extend this app to provide step tracking and add a suitable user interface. You should demonstrate your steptracking algorithm running in an Android app displaying a live step count whilst walking on level ground.

\*\*\* To test AS, BLE and app: use Nordic to find Characteristic for sensor, plug it in the code, look at the values. Plot the data.

# Embedded Development

This section concerns embedded development using the MBed embedded development platform. You will use this to receive sensor data and later to run your own step tracking algorithms.

Test firmware has been provided for you with the following functionality:

* Flashes an LED on the dev board
* Sends debugging information to the PC over serial
* Communicates with the Inertial Measurement Unit (IMU) breakout board
* Streams sensor data over Bluetooth Low Energy (BLE)

It is important to reproduce this behaviour before starting to modify the code running on the dev board, as it will rule out basic problems early on.

## NRF52-DK

You have been provided with a Nordic NRF52-DK board, which should be set up as follows:

### Updating the bootloader

1. Download the **0246\_sam3u2c\_mkit\_dk\_dongle\_nrf5x\_0x5000.bin** bootloader image.
2. Switch off the dev board.
3. Connect to your PC via USB.
4. Whilst holding down the rest button, turn on the dev board.
5. It should appear as a mass storage device.
6. Copy the bootloader image to the MSD.
7. Turn the board off and on again.
8. It should now be in a state where you can flash a program file.

### Flashing the board

First make sure that you can run a basic program on the dev board. You can always return to this example later to verify that your board is still working. The following firmware .hex files are available from the following GitHub repository:

1. Download the **BLE\_LED\_NRF52\_DK.hex** firmware image.
2. Switch on the dev board and connect it to your PC via USB. It should appear as a mass storage device.
3. Copy the firmware to the mass storage device.
4. Turn the board off and on again to run the program.
5. LED1 on the dev board should start to flash once per second.

If this example fails to run you may need to reflash the bootloader on your dev board, as described above.

## MPU-9250

The InvenSense MPU-9250 is an IMU motion tracking board containing a 3-axis accelerometer, gyroscope and magnetometer. These sensors can be used together to provide full 3D motion tracking.

### Wiring

You have been provided with an MPU-9250, which is mounted on a Sparkfun breakout board. This can be connected to the dev board using the I2C interface. You should make the following connections using jumper cables:

|  |  |  |
| --- | --- | --- |
| **Connection** | **Dev board** | **MPU breakout board** |
| Power | VDD | VDD |
| Ground | GND | GND |
| I2C Clock | P0.27 | SCL |
| I2C Data | P0.26 | SDA |

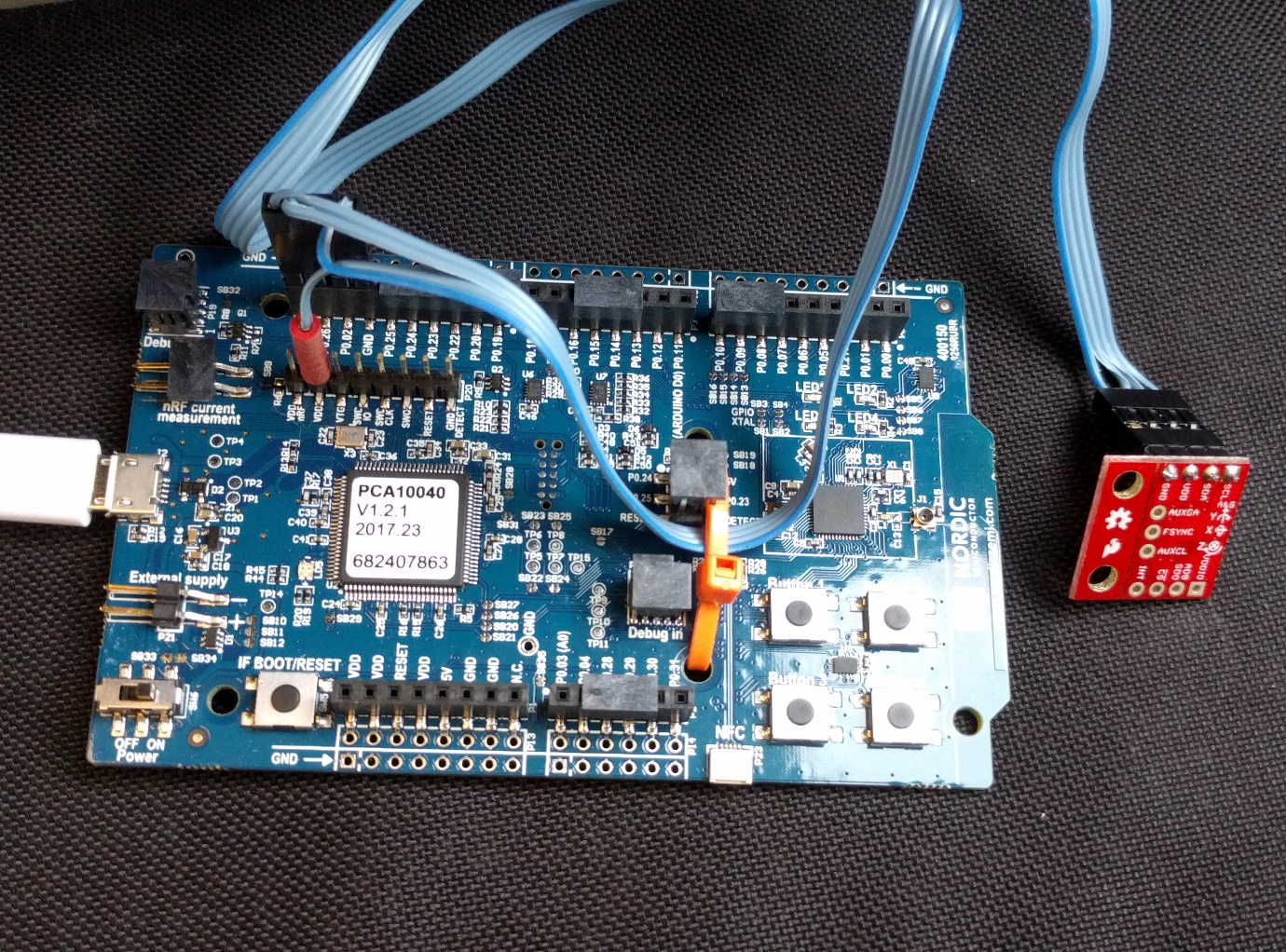


Figure 1: Connecting the MPU breakout board to the NRF52-DK

### Testing communication

The provided **I2C\_HelloWorld\_Mbed\_NRF52\_DK.hex** program tests communication between the NRF52-DK and the PC (via USB-serial port) and the MPU-9250 (via I2C).

1. Connect the dev board to the PC via USB and switch on.
2. Using a terminal application, open a connection to the J-Link CDC serial port using the following settings:
   1. Baud rate: 9600
   2. 8 bits
   3. Stop bits: 1
3. Copy the above .hex file to the NRF52-DK mass storage device.
4. The program should now run and display gyro output from the MPU. The values should be close to zero when static and increase when you rotate the sensor.

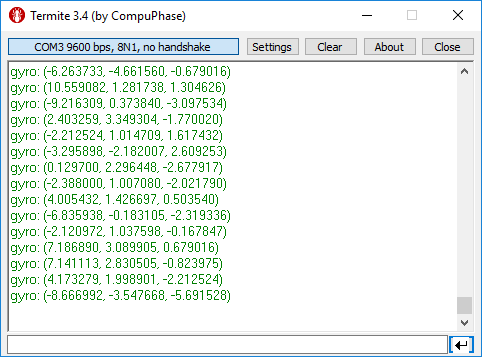


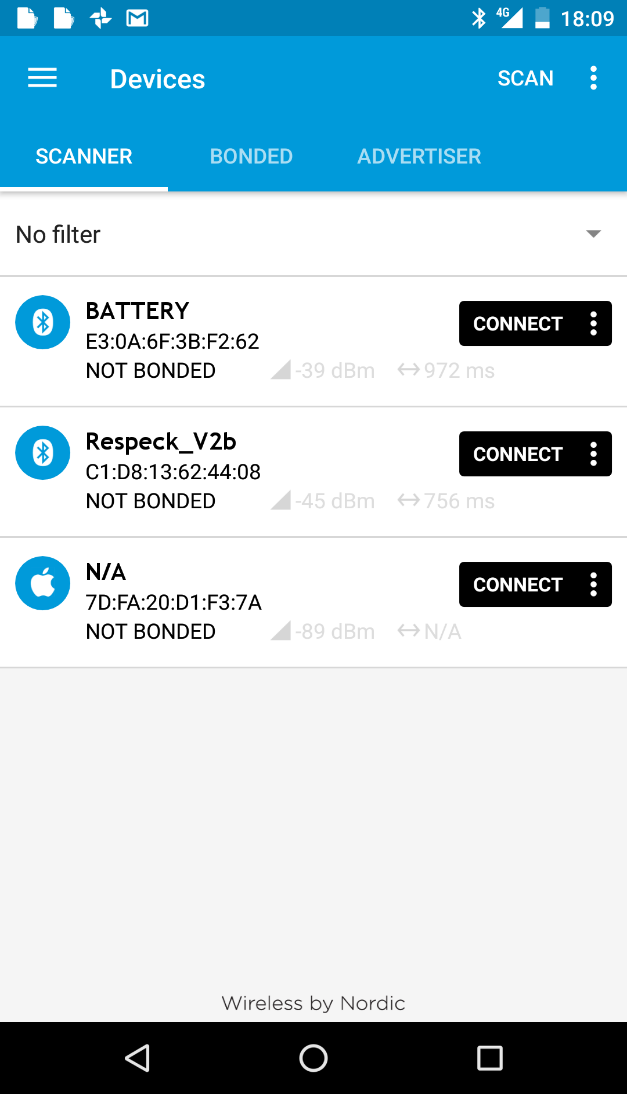
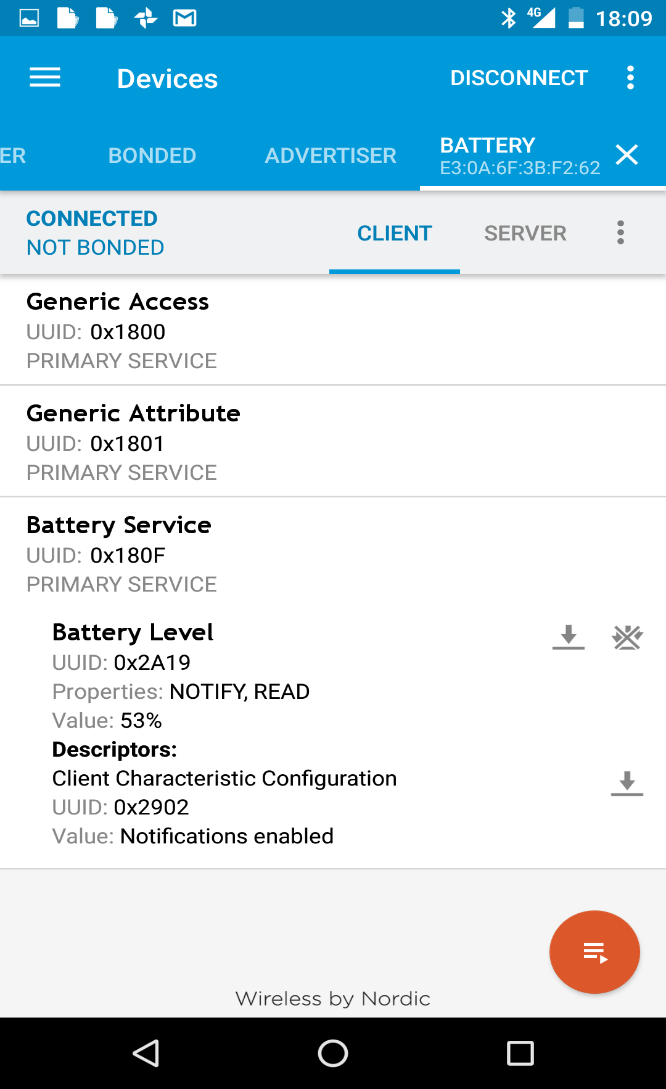
Figure 2: Expected serial output when testing MPU comms

### Testing BLE communication

We recommend using the excellent Nordic nRF Connect app for BLE debugging. This allows you to see the services provided by your BLE device, stream data from notifications and log it to a file. <https://www.nordicsemi.com/eng/Products/Nordic-mobile-Apps/nRF-Connect-for-mobile-previously-called-nRF-Master-Control-Panel>

Try loading the supplied **mbed-os-example-ble-BatteryLevel\_NRF52\_DK.hex** BLE example. This will send fake battery level values, for testing the Bluetooth communication.

1. Copy the .hex firmware image to your dev board as described earlier.
2. The dev board should now be discoverable over BLE, with the name “BATTERY”.
3. Open *nRF Connect* on your phone and scan for devices.
4. Press the *connect* button for the “BATTERY” device.
5. Expand the *Battery Service* and click the multiple down arrow icon next to *Battery Level* to stream battery percentage notifications.

## Mbed development platform

We will use the mbed development platform to compile firmware to run on the NEF52-DK board, which is fully supported in mbed OS 5. See the handbook for more information: <https://docs.mbed.com/docs/mbed-os-handbook/en/latest/>

### Online compiler

We recommend that you use the mbed online IDE and compiler for your firmware development.

Once you’ve registered for an account and set your hardware to the NRF52-DK you’ll have access to your own workspace. This will allow you to compile code and download the resulting firmware image to copy to the dev board. There is also an in-built version control system for you to use for your mbed projects.

### Example code

The OS5 mbed-os-exanple projects are a good starting point for your own firmware.

<https://os.mbed.com/teams/mbed-os-examples/code/>

We recommend starting with the blinky example, which flashes LED1 as seen in the first test program above.

<https://os.mbed.com/teams/mbed-os-examples/code/mbed-os-example-blinky/>

You can use the *Import into compiler* button to import the project into your own workspace in the mbed online compiler. Now try to compile and run the blinky example on your board.

### Local toolchain

Although it is possible to set up an embedded toolchain on your local machine, this can be tricky. You are free to attempt this at your own risk, but we won’t be providing any support!

### Libraries

There are several mbed libraries for communicating with the MPU-9250, which are available from the online compiler. The test firmware uses this one: <https://developer.mbed.org/teams/Edutech/code/MPU9250/>

### Debugging

There are several methods of debugging that you may find useful:

* LEDs – 4 of these can be switched on and off and is probably the simplest way to view output from your code
* Buttons – There are 4 buttons on the mbed board to you can set to perform actions in your firmware
* Serial output – As tested in the above example, provides more detailed output when connected to a PC
* BLE – More likely to be used for the final output of your firmware, but you can also send values for debugging
* If everything seems broken, please check the coin cell battery voltage on the board!

### Mbed bugs

Mbed is not perfect and you may experience compile errors or other bugs. Please share these and any solutions on piazza. Often rolling back the mbed-os library to the previous version using the *Revisions* option in the online compiler will fix build errors.