Tools for smartphone multi-sensor data registration and GT mapping for positioning applications

Antonio R. Jiménez*, Fernando Seco* and Joaquín Torres-Sospedra†

*Centre for Automation and Robotics (CAR) CSIC-UPM, 28500 La Poveda, Arganda del Rey, Madrid, Spain
†Institute of New Imaging Technologies, Universitat Jaume I, Avda. Vicente Sos Baynat S/N, 12071 Castellón, Spain
antonio.jimenez@csic.es – fernando.seco@csic.es – jtorres@uji.es – http://www.car.upm-csic.es/lopsi – http://geotec.uji.es/

Abstract—Nowadays smartphones have impressive sensing and computation capabilities, allowing the registration and processing of multiple sources of information. This power enables the creation of useful applications, such as seamless location both outdoors and indoors. Research teams pay less interest in standardizing the acquisition and processing of sensor data than to research and innovation tasks, so each group develops its own private software tools to collect data. We want to contribute by creating a framework that allows a more coherent data-stream registration and algorithm performance comparison. In this paper we present an open-source framework to make possible the use of a common data format, creating a de-facto standard format. It provides the data parsers able to interpret and visualize each specific data type. The framework also has a graphical user interface (GUI) to geo-reference and calibrate indoor maps. With this GUI tool, it is possible to define ground-truth (GT) trajectories as a sequence of reference points on the map. The subsequent performance comparison of positioning algorithms requires a good ground-truth [6].

In section, II, we present the GetSensorData Android app, describing the different internal and external sensors that can be registered, the format for each sensor type, and the data parsers. In section III, we present the mapping user interface to geo-calibrate indoor maps and to define ground-truth paths. Finally, section IV shows the usage of these tools.

II. SMARTPHONE-BASED MULTISENSOR REGISTRATION

A. The GetSensorData App

We have designed an Android-based App that is able to show in the main screen the different sensors available in a smartphone. The sensors detected in the phone’s hardware are marked in green color, whereas the common sensors not detected in the phone’s hardware are marked in red color (see left screenshot in Fig. 1). In this way, it is very easy to visualize the richness of sensors available in the phone.

Three different buttons are available in the top of the main screen. The left button (“Show Sensor Features”) opens a parameter description for each sensor. Information such as manufacturer, model version, resolution, maximum range, power consumption or sampling rate is shown (see gray areas in the right screenshot of Fig. 1).

The button in the middle (“Show Real-time Data”), opens the real-time display for each of the available sensors (see white areas in Fig. 1 right). The sampling frequency is estimated by the application.

The sampling frequency depends on the sensor’s hardware, but it can be configured to different values using the configuration screen (Fig. 2 left). There are two ways to configure the sampling rate. First, by choosing between 4 different modes available at the Android sensor API (FASTEST: that get sensor data as fast as possible, GAME: rate suitable for games, NORMAL: default rate suitable for screen orientation changes, and USERINT: rate suitable for the user interface). See options...
Fig. 1. Main screen of GetSensorData Android app. Left: initial mode showing all potential sensors (green: available; red: not available or not connected). Right: visualization of real-time sensor data and static sensor features (this screenshot shows triaxial accelerometers and gyroscopes); other sensors can be visualized by scrolling down the view.

at Fig. 2 right. They are only a hint to the system, and do not generate any systematic way to fix the sample rate.

Another way to set the desired sampling frequency is by directly typing it. This is possible if the parameter “MinDelay", in the sensor description, is not equal to zero. If a sensor returns a “MinDelay” of zero, it means that it is not a streaming sensor because it reports data only when there is a change in the parameters it is sensing. Streaming sensors (“MinDelay” non zero) are preferred since they sense data at regular intervals. A value of “MinDelay” such as 10,000 (as appears in the accelerometer description of Fig. 1 right) should mean that the minimum sampling interval is 10,000 µs (i.e. a maximum 100 Hz sampling rate). A value of 5,000 implies a maximum sampling rate of 200 Hz. The desired frequency within the valid range, can be configured in the configuration window (Fig. 2 left), but it is also a hint to the system; there is no guarantee of correctly setting it up.

The button at the right (“Start Saving a LogFile”), in Fig. 2, triggers the recording of all sensing data in a logfile for later off-line use. This logfile is stored in folder "Logs Files GetSensorData" with a unique name based on the date and time at the end of the recording. The logfile will be containing all the available signals captured in real-time, which could include: WiFi and Bluetooth received signal strength (RSS), inertial data, GNSS positions, etc. Additionally, the utility of the software can be augmented by connecting external sensors that stream data into the smartphone. Currently, this extension is limited to those sensors used in our CAR center for positioning research, such as RFID readers and inertial measurement units. The number of external sensors, could be increased by collaborative programming.

B. Internal sensor sources

The internal sensors are those defined in the Android developer sensor overview guide1. There are two types of sensors in Android: hardware and software-based sensors. Hardware-based sensors are physical components built into the phone, and data is obtained directly from them. On the other hand, software-based sensors are algorithms that, using some readings from one or several hardware-sensors, emulate a new sensor type (for example: gravity, step counter, orientation).

The app mainly registers hardware-based sensor such as WiFi RSS, Bluetooth RSS, inertial data (accelerometer and gyroscope), magnetic field, atmospheric pressure, which are directly usable for positioning, and also signals of opportunity, such as, temperature, humidity, proximity and light intensity, that could be used to give clues about the user’s location (e.g., the light-matching approach [7], [8]). It also registers software-based sensor such as phone’s orientation, as well as sound intensity that is estimated from the microphone readings.

Additionally, we include geo-spatial coordinate estimations, derived form the network (Cell telephony or WiFi fingerprinting) or from GNSS integration. The latter is available when precise positioning is activated in the phone and visibility to satellites is good. Fig. 3 shows a screenshot during geo-spatial positioning using NETWORK (left) and GNSS (right). Note that network positioning only performs horizontal 2D location, and does not estimate the altitude. When there are enough satellites “in view” with good quality, then the estimation switches to GNSS positioning mode, and satellites used in the trilateration are marked as “in use”.

At this state the registration of the RAW measurements from multi-contellation GNSS systems, which a trending topic [9], is not implemented yet, but hopefully soon.

1https://developer.android.com/guide/topics/sensors/sensors_overview#java
C. Logfile data format

The logfile is a “txt” file containing multiple rows. The continuous stream of data generated in the phone is stored in the logfile in sequence, row by row as they are received. Each row begins with a unique identification header, composed by four capital letters followed by a semicolon (e.g. ‘WIFI’, ‘ACCE’, ‘MAGN’, ‘BLE4’, ‘GNSS’, etc.). This header determines the kind of sensor. The subsequent data fields after the header, separated by semicolons (“;”), contain different sensor readings of interest, including timestamps. In Fig. 4 an extract of a real log file is shown as example.

Fig. 3. Screenshots with the location service. Left: Network location mode using WiFi or cell phone trilateration. Right: GNSS localization with higher accuracy and height estimation.

Fig. 4. Logfile sample of a sequential sensor data registration, as created with the GetSensorData app.

The tri-axial accelerometer (ACCE) and gyroscope (GYRO) rows contain two time stamps followed by the X, Y, and Z components. The inertial data is registered this way:

- **ACCE**: Phone’s tri-axial acceleration in \( \text{m s}^{-2} \);
- **Format**: “ACCE; AppTimestamp(s); SensorTimestamp(s); Acc_X(m s\(^{-2}\)); Acc_Y(m s\(^{-2}\)); Acc_Z(m s\(^{-2}\)); Accuracy(int)”
- **Example**: “ACCE;0.034;8902.708;−1.800446;7.00465;7.173033;3
- **GYRO**: Phone’s turn rate in \( \text{rad s}^{-1} \);
- **Format**: “GYRO;AppTimestamp(s); SensorTimestamp(s); Gyr_X(rad s\(^{-1}\)); Gyr_Y(rad s\(^{-1}\)); Gyr_Z(rad s\(^{-1}\)); Accuracy(int)”
- **Example**: “GYRO;0.032;8902.705;−0.22846;−0.21930;−0.054983;8

The first timestamp (“AppTimestamp”) is set by the mobile App when data is read at the listener (interrupt processing routine). This timestamp may not be totally representative of when data was actually captured by the sensor, but it defines a common time reference for all sensors. On the other hand, an additional not-always-available time stamp, called “SensorTimestamp” is set by the sensor itself and can be more representative of the actual time between consecutive samples. The sampling interval is in fact the difference between two consecutive samples SensorTimestamp\((k)\) and SensorTimestamp\((k-1)\).

Note that the sampling rate of each sensor type can be different from logfile to logfile, specially when using different phones, since it depends on the embedded sensor chipset in a particular phone. Typical sampling frequency values for the inertial data is about 50 to 200 Hz. Pressure, Sound, Light sensors have a much lower update rate (< 10 Hz). WiFi scans are available approximately every 4 to 6 s (≈0.16 to 0.25 Hz).

The triaxial magnetometer (MAGN), with sampling rates similar to those of accelerometers, are recorded in this format:

- **MAGN**: tri-axial magnetic field in \( \text{µT} \);
- **Format**: “MAGN; AppTimestamp(s); SensorTimestamp(s); Mag_X(µT); Mag_Y(µT); Mag_Z(µT); Accuracy(int)”
- **Example**: “MAGN;0.035;8902.708;−20.7000;−34.0200;−19.2000;3

An Android software sensor, which estimates the orientation of the phone, is created from the inertial and magnetometer data. Information is coded in rows starting with an AHRS header (AHRS stands for Attitude and Heading Reference System). The rotation conventions are described in the Android developer guides, see Sensor Coordinate System section in the sensors overview guide\(^1\) and Use the rotation vector sensor section in the sensors motion guide\(^2\).

- **AHRS**: the mobile phone 3D orientation in terms of pitch, roll and yaw, and also as a rotation vector (a 3-value compressed form of a quaternion):
- **Format**: “AHRS; AppTS(s); SensorTS(s); PitchX(°); RollY(°); YawZ(°); RotVecX(°); RotVecY(°); RotVecZ(°); Accuracy(int)”
- **Example**: “AHRS:0.033;8902.705;4.6550;11.7495;−124.0558;0.25038;−0.26750;−0.80468;−2”

\(^1\)https://developer.android.com/guide/topics/sensors/sensors_overview
\(^2\)https://developer.android.com/guide/topics/sensors/sensors_motion
The WiFi RSS data, as well as BLE RSS, are important sources of information for absolute positioning indoors. They can be used to implement fingerprinting or trilateration localization methods. We register WiFi RSS information this way:

- **WIFI**: RSS (in dBm) received from a particular AP
  - **Format**: “WIFI; AppTimestamp(s); SensorTimeStmp(s); NameSSID(text); MACBSSID(text); Frequency(Hz); RSS(dBm)”
  - **Example**: “WIFI:16:62:22:52:eduroam:00:0b:86:27:3c:d0; 2427;−75”

In order to register BLE4 beacons, we must set the checkbox in the App’s configuration window (see Fig. 2 left) and restart the app (BLE4 line should appear now green, as in Fig. 1 left).

Two different types of beacons can be registered: iBeacon and Eddystone (the dominant protocols from Apple and Google, respectively). As both protocols have different fields and identification codes, we use two different line formats, but with common fields. The format for both BLE4 lines are:

- **BLE4**: RSS (dBm) from iBeacons tags
  - **Format**: “BLE4; AppTimestamp(s); ‘iBeacon’; MAC(text); RSSI(dBm); TxPower(dBm); MajorID(int); MinorID(int); UUID(text)”
  - **Example**: “BLE4:0.156;iBeacon;FE:5E:61:D4:F2:EC;−69;−76; 2016:19:9407;f30-f5f8-466-aff9-2556657fe6d”

- **BLE4**: RSS (dBm) from Eddystone tags
  - **Format**: “BLE4; AppTimestamp(s); ‘Eddystone’; MAC(text); RSSI(dBm); instanceID(int); OptionalTelemetry[voltage(V); temperature(°C); uptime(ms); count(int)]”
  - **Example**: “BLE4:0.154:Eddystone;E5:A3:7B:5D:3E:9A;−98; 201600000000010;5954;17.0;20240;13591778”

Pressure and light data, can give some clues about potential floor or room changes. They are registered with the PRES and LIGH headers with following formats:

- **PRES**: the atmospheric pressure in mbar
  - **Format**: “PRES; AppTimestamp(s); SensorTimestamp(s); Pres(mbar); Accuracy(int)”
  - **Example**: “PRES:0.038:8902.726:956.4289:0”

- **LIGH**: for light intensity in lx
  - **Format**: “LIGH; AppTimestamp(s); SensorTimestamp(s); Light(lx); Accuracy(int)”
  - **Example**: “LIGH:0.032:8902.693:292.0:0”

The GNSS position information is recorded with latitude and longitude, height, bearing, speed and the number of satellites in view and in use.

- **GNSS**: On-chip fused GNSS estimation of location
  - **Format**: “GNSS; AppTimestamp(s); Lat(°); Lon(°); Altitude(m); Bearing(°); Accuracy(m); Speed(m s−1); UTCTime(ms); SatNumView(int); SatNumInUse(int)”
  - **Example**: “GNSS:0.611:40.313524;−3.483137;600.865.0000; 4.0;0;0;1358782729999;17:15”

Additional internal sensors such as ambient temperature, sound, proximity to the phone and relative humidity are easy to interpret from the header’s information at the top of each logfile.

### D. External sensor sources and format

The GetSensorData app can read data from some external sensors. Three different inertial sensor units, showed in Fig. 5, can be read: MTi and MTi-G from Xsens⁴, LPMS-B from Life Performance Research⁵, and Osmium MIMU22BT from InertialElements⁶.

The MTi and MTi-G Xsens sensors must be connected to the phone through a USB On-The-Go (OTG) cable, so the smartphone has to support USB OTG protocols. The GetSensorData app automatically detects the Xsens device and sets a default configuration with a 100 Hz sampling rate.

The other two IMUs LPMS and MIMU can be connected by Bluetooth 3.0 wireless protocol. You must pair them prior to open the GetSensorData app. After pairing, when the app runs, it initiates a hidden scan process in order to establish communication with the sensors. When communication starts, the app marks that external sensor in green color, and the system is ready to visualize data and to register it into a logfile. The Osmium MIMU also can be connected by USB cable.

The Xsens Mti device is the sensor that we have used the most in our experimentation. It is a ruggedized small MEMS inertial measurement unit with an embedded processor capable of calculating in real time the attitude (roll, pitch and yaw; or in quaternion format), as well as outputting calibrated 3D acceleration, rate of turn (gyro) and (earth) magnetic field data. We use string ’IMUX’ as line header.

- **IMUX**: Xsens inertial data
  - **Format**: “IMUX; AppTimestamp(s); SensorTimestamp(s); Counter(int); Acc X(m s−2); Acc Y(m s−2); Acc Z(m s−2); Gyr X(rad s−1); Gyr Y(rad s−1); Gyr Z(rad s−1); Mag X(μT); Mag Y(μT); Mag Z(μT); Roll(°); Pitch(°); Yaw(°); 4 x Quat(float); Pressure(mbar); Temp(°C)”
  - **Example**: “IMUX:0.185:28.58:2874.0:0.0311;−0.0217:9.8104; 0.0166:0.025;0.0162:0.3943:0.2547;−0.6813;0.0000:0.0000; 0.0000:9.5922;0.0000:0.00004;−0.2824; 2000:24.25”

The IMUX data includes a counter to check if any sample of data is lost. An atmospheric pressure field is filled when MTi-G sensor is connected. Registering the internal temperature of the sensor is valuable to check if it is stable. The temperature stabilizes and drift minimizes after operating 15 minutes.

---

⁴https://www.xsens.com
⁵https://lp-research.com/lpms-b/
⁶https://inertialelements.com
On the other hand, the LPMS inertial sensor has similar functionality to Xsens, but is smaller and has Bluetooth wireless connectivity, as main advantages. The chosen sensor header is ‘IMUL’. The recorded data fields are the same as in the Xsens (IMUX) case.

The new compatible IMU is Osmium MIMU-BT (header ‘IMUI’). This model uses Bluetooth 3.0 protocol (as LPMS). This inertial sensor has the advantage of including an array of four 9-axis IMUs, so the inertial signals, after an embedded fusion, are less noisy and consequently the bias and drift can be improved. The device also includes a step detection and PDR algorithm for accurate foot-mounted estimation. Multiple output configuration are possible, including raw inertial signals, but we opted to output the delta displacement at each detected step (displacements in X, Y, Z and heading with respect to the last foot position). This information is enough to reconstruct the trajectory of a walking person. The estimate covariance matrix (4x4) is provided in compact format (10 numbers) as it is symmetrical.

- **IMUI**: Osmium MIMU-BT inertial and PDR sensor data
  - **Format**: “IMUI; AppTimestamp(int); Packet_Count(int); Step_Counter(int); delta_X(m); delta_Y(m); delta_Z(m); delta_theta(°); 10 x Compact Covariance (float)”
  - **Example**: “IMUI:22.70;22:18:0.57450;−0.55372;0.05466;1.00623;0.00038;0.00000;−0.00006;−0.00000;0.00038;−0.00000;−0.00000;0.00037;−0.00000;0.00000”

The “Packet_count” field is an internal MIMU counter with “Packet_count” increments less than “Packet_count” because each data packet corresponds to a new generated step data. If “Step.Counter” increments less than “Packet_count” it means that information is lost during Bluetooth communication (not so uncommon in our tests).

At this stage, we are still trying to find the best configuration to make this communication as robust as possible, even by testing other more recent model: Osmium MIMU22BLPX (in this case using Bluetooth v4.1 and USB 2.0 data interfaces).

Apart from inertial sensors, the GetSensorData app is able to connect with RFID readers, in particular model M220 from RFCode3. This model is the one used in many of our past research works about RFID indoor positioning and tomography. Fig. 6 shows a M220 RFID reader and some external inertial sensors connected to a smartphone running the GetSensorData app. Note that correct communication is marked in green color in the App’s main screen.

It is important to mention that we can connect unlimited number of RFID readers to the app, since on start-up the app tries to detect and connect to all compatible paired Bluetooth devices. In the logfile (RFID lines), each reader is identified by its unique Reader Number.

- **RFID**: Portable wireless M220 RFID reader from RFCode
  - **Format**: “RFID; AppTimestamp(s); ReaderNumber(int); TagID(int); RSS_A(dBm); RSS_B(dBm)”
  - **Example**: “RFID:0.187;30:67955:64;55”

Using the Received Signal strength (RSS) between the emitting ‘TagID’ and the received ‘ReaderNumber’ trilateration or fingerprinting location solutions can be implemented. As the reader has two antennas (A and B) a redundant, but useful, RSS information is obtained, which can be used to make estimations more robust, for example by selecting the strongest RSS value (less affected by NLOS).

**E. Extra functionalities: Ground-truth marks and Parsing tool**

The app has a button (down on the main screen, called “Mark First Position”) that is intended to insert ground-truth (GT) reference lines (header ‘POSI’). This is useful when you latter want to relate the logfile signals with some key events (usually ground-truth positions), but also to mark the occurrence of other events such as activities (sit down, stop moving, etc.). The POSI lines have the following format:

- **POSI Reference for ground-truth position**
  - **Format**: “POSI; Timestamp(s); Counter(int); Lat(°); Longitude(°); floor_ID(int); Building_ID(int)”
  - **Example**: “POSI:0.033;4;...”

The ‘POSI’ line initially is created with no ground-truth (GT) data, as no GT is apriori known, and the app does not ask you to enter that data during the recording process. The GT information can be inserted a posteriori using a geo-referenced mapping tool running in Matlab that is presented in next section (sec. III).

The parsing of the all internal and external sensor readings in the logfiles can be done easily using a Matlab function, called ReadLogFile21.m8. This function reads line-by-line all sensor streams, and load them on Matlab numerical matrices, which are saved as *.mat files for a more efficient later use. The parser function also open several windows to visualize data graphically. In Fig. 7 a few data examples are shown for turn rate, BLE RSS and GNSS position estimations, while a person is moving and rotating the phone.

---

3https://www.rfcode.com/

8Available at http://car.upm-csic.es/lopsi
Common available floor maps for buildings are typically based on engineering CAD formats. These maps are not usually geo-referenced, so it is not easy to use them in navigation applications where a mobile device has information about its location using a globally referenced coordinate frame such as WGS84 (the one used by GPS). Having in mind the application goals we already defined (competitions, teaching or collaborative works among researchers), we considered useful to create a graphical tool in order to easily and systematically calibrate maps, create ground-truth points and to test the performance of different positioning algorithms.

In the next subsections we present this graphical tool called ‘GUI_maps.m’ that has been created using Matlab code. This tool can be used to: 1) geo-reference indoor maps of a building (several bitmaps for each different floor); 2) add sequences of points of ground-truth marks; and 3) to manipulate the sensorial LogFiles created with the GetSensorData Android app in order to fill, or empty, the ground-truth ‘POSI’ marks inside those Logfiles.

### A. User Interface

The ‘GUI_maps.m’ tool has a simple screen interface as can be seen in Fig. 8, with six different actuation areas (marked in red numbers):

1. **Area 1**: Background georeferenced Google Maps with longitude and latitude information.
2. **Area 2**: Control to select the desired type of view to appear in area 1: terrestrial or satellite.
3. **Area 3**: Used to calibrate several CAD indoor bitmaps representing floorplans in a building.
4. **Area 4**: Creation of ground-truth paths (list of geo-points) with indication of the Latitude, Longitude, FloorID and BuildingID.
5. **Area 5**: Used to manipulate the empty ‘POSI’ marks in LogFiles created with the GetSensorData Android app.
6. **Area 6**: Contains controls for zooming in and out, dragging, and measuring distances.

### B. Map Calibration

For map calibration, we need to have the bitmap files with enough resolution to capture the indoor details. In current version only jpg images are supported.

The tool does not yet include the initial creation of the calibration file, so it is required to manually create an approximate calibration file (this process is required to be done only once).

In Fig. 9 a calibration file example is shown containing 4 floors (IDs: 3, 2, 1 and 0) and one building (ID: 1).

The calibration file is a text file with extension ‘.cal’. Each floor in the calibration file must have seven rows:

1. the filename of the bitmap
2. the floor number or ID (e.g. -2, 0, 3)
3. the Building number or ID (e.g. 10, 20, 30)
4. Latitude (in degrees) of image center
Fig. 8. Main Matlab GUI interface with the different controls to calibrate maps, define GT trajectories, and to update logfiles with POSI marks.

Fig. 9. Sample of a map calibration file (text file with ‘cal’ extension)

5) Longitude (in degrees) of image center
6) Clockwise floormap image rotation (in degrees) for correct alignment with background (geometric north)
7) Scale of floormap image pixels (meters/pixel)

A calibration file has always 7 lines, and a variable number of columns; as many columns as the number of floors in the building (in the example in Fig. 9 there are 4 columns for each of the 4 floors).

Once the approximate calibration file is created in a text editor, we can open it and refine the file using the scaling, rotation and displacement controls in area 3 (Fig. 10). The calibration is achieved by fitting as best as possible the bitmap over the Google maps in satellite or terrestrial view. In order to ease the overlapping of both images (indoor with respect the Google maps) we can change the transparency of the bitmap.

Fig. 10. Calibration process using the map manipulation controls

In order to reduce the uncertainty in the calibration it is recommended to check the calibration of the scale of the map using a distance meter device (laser meter or a long metric tape). We can physically measure the length of some corridors or halls in the real space and then compare that real distance with the distance estimated with the graphical distance tool.

To refine the translation and rotation of the floor bitmap, it is recommended to measure the real distance (laser meter) between some marks in the outdoor terrain with respect to some outer walls in the building. With this low-resource method we believe it is possible to calibrate the maps with an accuracy better than 0.5 meters. If bitmap already contains geo-referenced points (at least 3) the calibration is straightforward and more accurate.

C. Ground-truth point creation

Once the map is calibrated, now it is possible to create the list of ground-truth (GT) or geo-referenced points. The control buttons in area 4, as shown in Fig. 11, should be used.

These points are stored in a text file in a trajectory format (with extension ‘*.tra’). These trajectories GT points can be opened to interactively modify their positions, or manipulated by inserting or deleting some of them.

D. Ground-truth management in POSI marks

The tools in area 5 are used to integrate the GT points created before (a ‘*.tra’ file) with the sensor logFile corresponding to a sensor data registration experiment. When opening the logFile, it starts parsing the file. After completion, the views in area 5 will show the number of lines in the log files and the number of POSI marks found, distinguishing between empty and filled ones (i.e. those with a correct GT).

By pressing the ‘Assign Trajectory as Ground-Truth POSI to LogFile’ button the integration of both sources of information is done. After saving, a new LogFile ending with ‘*.POSI_assigned.txt’ is created, which has the POSI lines with the correct GT; e.g., ‘POSI:0.033:4.40.31347; -3.48315;0.10’.

It is also possible to completely delete POSI marks from logFiles (those ended in ‘*.POSI_deleted.txt’).
IV. USE OF TOOLS IN LOCALIZATION ACTIVITIES

The previous sections presented tools (GetSensorData, the parser, and the mapping tool with geo-GT management), that have several uses in the indoor positioning community. We encourage people to use these free tools (available in our website\(^9\)) to make their own mobile phone experiments, and then to share them with the indoor community, as a way to create a comprehensive repository of experimentation databases. Also the source code is available at a Git repository\(^9\), so that the community can contribute to improve it.

Apart from the use for smartphone data registration in experiments for research, other uses are possible. Some of these will be presented next.

A. Indoor localization competitions

During the last editions of the Indoor Positioning and Indoor Navigation (IPIN) Conference, years 2016, 2017 and 2018, we have used versions of these tools in the localization competition: Track 3 - ‘Smartphone based Positioning (off-site)’. The details of the training and evaluation logfiles are available in the corresponding IPIN conference websites.\(^{10,11}\) The challenges, competitor method details, and results have been reported in scientific publications\(^4,10\).

The provided logfiles and maps have become familiar to several competitors that participated in past editions\(^11,12,13\). The format is not a standard but it is a reasonably way to share experiments between researchers. So far, several different buildings have already been tested (four research buildings in Spain and one shopping mall in France)\(^14\).

B. Teaching PDR algorithms

The tools have also been used in teaching activities. For example, the PDR tutorial presented at IPIN 2017 edition in Japan, had a practical session where attendees were requested to use their smartphone with the GetSensorData app, in order to make experiments. In this way, they could put hands on Matlab, or Octave, environment to better learn the PDR concepts and methods presented at the tutorial. Other new editions of tutorials are planned to use these tools as a quick way to record live data, visualize and better learn the estimation methodologies.

V. CONCLUSIONS AND FUTURE IMPROVEMENTS

In this paper we have presented a range of tools that are useful to the research community in indoor/outdoor localization with smartphones. The GetSensorData app is able to register signals from diverse sensors at their update rates. New external sensors can be added in order to centralize the recording under a common lightweight device with a common time reference. Several Matlab tools have been designed to parse sensor information and also to generate ground-truth points with the support of calibrated floor maps provided as bitmaps.

Although there are some attempts to provide free and open tools to support indoor localization, this is the first stand-alone open-source multi-sensor framework for smartphone indoor positioning as far as we know.

Currently, the access to raw sensor data from multiple GNSS constellations and frequencies is a hot topic. We plan as a future work to integrate that pseudorange information into new headed lines (‘GRAW’). Also, we would like to encourage the research community to contribute to the improvement and addition of new functionalities in our Git repository\(^9\).

ACKNOWLEDGMENT

The authors thank the financial support received from projects: TARSISU (TIN2015-71564-C4-2-R), REPIN Plus (TEC2017-90808-REDT), Tomoloc (PIE201750E003), TECHNOFU\-SION-3-CM (S2018/EMT-4437) and MICROCEBUS (RTI2018-095168-B-C55).

REFERENCES