SoftwareX 19 (2022) 101186

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

GetSensorData: An extensible Android-based application for multi-sensor data registration

Juan D. Gutiérrez [a,*], Antonio R. Jiménez [b], Fernando Seco [b], Fernando J. Álvarez [a], Teodoro Aguilera [a], Joaquín Torres-Sospedra [c], Fran Melchor [d]

[a] Sensory Systems Research Group (GISS). Universidad de Extremadura, Av. de Elvas s/n, 06006 Badajoz, Spain
[b] Center for Automation and Robotics, CSIC-UPM Ctra. Campo Real km 0.2, 28500 Arganda del Rey, Spain
[c] Centro ALGORITMI, Universidade do Minho, Alameda da Universidade, 4800-058 Guimarães, Portugal
[d] Quercus Research Group, Universidad de Extremadura, Avenida de la Universidad s/n, 10003, Cáceres, Spain

ARTICLE INFO

Article history:
Received 3 May 2022
Received in revised form 23 July 2022
Accepted 29 July 2022

Keywords:
Sensing technologies
Mobile applications
Android

ABSTRACT

Smartphones are powerful tools with extensive sensorization that can provide useful information in research or everyday life applications. This information can be obtained from the device's built-in sensors or through other external sensors connected physically via USB or wirelessly via Bluetooth or WiFi. This paper presents the GetSensorData application that provides an open-source, flexible and extensible framework for registering sensor data from Android devices. The application uses standard formatting and synchronization, easing interoperability with other software. End developers (particularly those involved in research) can save the effort and time of creating their sensor acquisition applications and fully concentrate on the higher-level data processing tasks. The application has been used and successfully evaluated for six years by various research groups in different activities related to their work areas. Some examples are the calibration of positioning systems in competitions held at conferences, modeling wireless signal path loss propagation in indoor environments or data collection for unsupervised learning algorithms.

Code metadata

Current code version	2.3.1
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-22-00113
Code Ocean compute capsule	
Legal Code License	GNU General Public License v3.0
Code versioning system used	Git
Software code languages, tools, and services used	Java, Kotlin
Compilation requirements, operating environments & dependencies	Android Studio 2021.1.1 Patch 3, Android 5.0 (Lollipop, API 21)
If available Link to developer documentation/manual	https://gitlab.com/getsensordatasuite/getsensordata_documentation
Support email for questions	lopsi.csic@gmail.com

Software metadata

Current software version	2.3.1
Permanent link to executables of this version	https://gitlab.com/getsensordatasuite/getsensordata_android/-/blob/master/Releases/GetSensorData-2.3.1.apk
Legal Software License	GNU General Public License v3.0
Computing platforms/Operating Systems	Android 5.0 (Lollipop, API 21)
Installation requirements & dependencies	
If available, link to user manual - if formally published include a reference to the publication in the reference list	https://gitlab.com/getsensordatasuite/getsensordata_documentation
Support email for questions	lopsi.csic@gmail.com

* Corresponding author.
E-mail address: andy@unex.es (Juan D. Gutiérrez).

https://doi.org/10.1016/j.softx.2022.101186

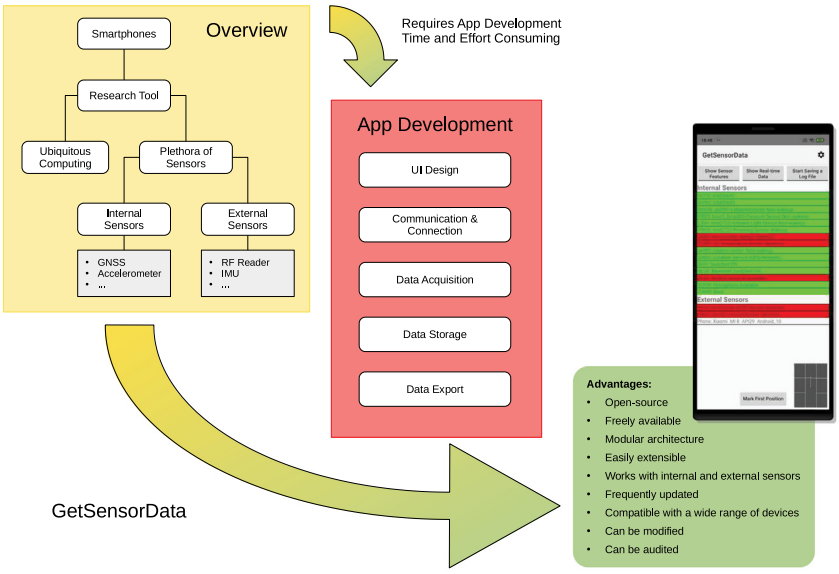


Fig. 1. Proposed solution for the use of smartphones as research tools.

1. Motivation and significance

The processing capabilities of modern smartphones allow real-time recording of multiple sensory data streams from their built-in and externally connected sensors. This processing capacity enables more complex applications such as improved indoor and outdoor positioning. However, each research group acquires, pre-processes and stores sensor data in the way that better suits their objectives. This lack of consistency in format makes it difficult to compare results and share data, and also weakens the capacity for joint efforts between researchers from different groups [1,2]. In research works, the method for acquiring raw sensor data, its formatting or pre-processing, as well as the data itself, are not always shared. Fortunately, this trend is changing, and currently, it is easier to find datasets and supporting material shared by authors [3,4]. Even though there are software tools to develop applications for portable devices, the creation and integration of such modules can be a daunting task for non-experts, spending time and resources to develop applications for sensory data acquisition.

There are a wide variety of applications to capture and save sensory data on Android devices. The following is a selection of the most relevant applications with these functionalities. Sensor Data.[1] is the top result in the Android App Store when looking for sensor data applications, with more than a hundred thousand downloads. This application can capture data from the built-in sensors of an Android device for later analysis. Although it is offered for free, some of its features are locked behind an in-app purchase, and its source code is not available. With a similar set of features, Sensor Data Logger[2] also offers its source code. Unfortunately, this application has not been updated for years, which may result in the application ceasing to work as proper support is not provided. Finally, OpenCamera Sensors[3] shows more activity than the previous two ones and also offers its source code. However, the number of available sensors is limited to the camera and Inertial Measurement Units (IMUs) In this case, the actual camera acts as the primary sensor while the rest of the sensory data are added to the captured images. Unfortunately, none of these applications allows the use of external sensors. More importantly, none offer a modular architecture that allows researchers to add their sensors, either internal ones not supported by the application or external ones connected by cable or wirelessly. The ideal application for collecting sensory data should be updated frequently and be compatible with a wide range of devices. Its code should be open-sourced so that it can be modified and audited. A qualified developer team should back its maintenance. It should work with built-in and external sensors, be easily extensible, and be freely available.

This paper introduces GetSensorData, an Android application focused on sensory data acquisition. The data can be displayed in real-time on the device screen, or saved for later analysis, creating a de-facto standard format. The application is open-source, and its development is community-driven via Git pull requests. Its architecture is designed to foster collaboration, encouraging others to include the sensory software needed in the application and making it available for the research community. GetSensorData features a modular and extensible class hierarchy for sensor management, and it is accompanied by a suite of tools to perform tasks such as parsing or visualization of acquired data in a personal computer. A diagram describing the benefits of this application is depicted in Fig. 1.

The application was initially conceived to help in the development and evaluation of positioning algorithms. It has readily been used in different indoor positioning competitions since the 2016 International Conference on Indoor Positioning and Navigation (IPIN) [2,5–7]. For this purpose, a specific Graphical User Interface (GUI) to accurately geo-reference and calibrate indoor maps has been included in the GetSensorData suite. This tool also assists in defining Ground-Truth (GT) trajectories as a sequence of reference points on the map. Starting with the first version of GetSensorData, the application has been tested and used by many scientific groups to improve their positioning algorithms. However, GetSensorData is not constrained to localization tasks: researchers interested in harvesting information from sensory data can use and expand the application to cover their own needs.

[1] https://play.google.com/store/apps/details?id=com.matlabgeeks.gaitanalyzer

[2] https://play.google.com/store/apps/details?id=net.steppschuh.sensordatalogger

[3] https://github.com/MobileRoboticsSkoltech/OpenCamera-Sensors

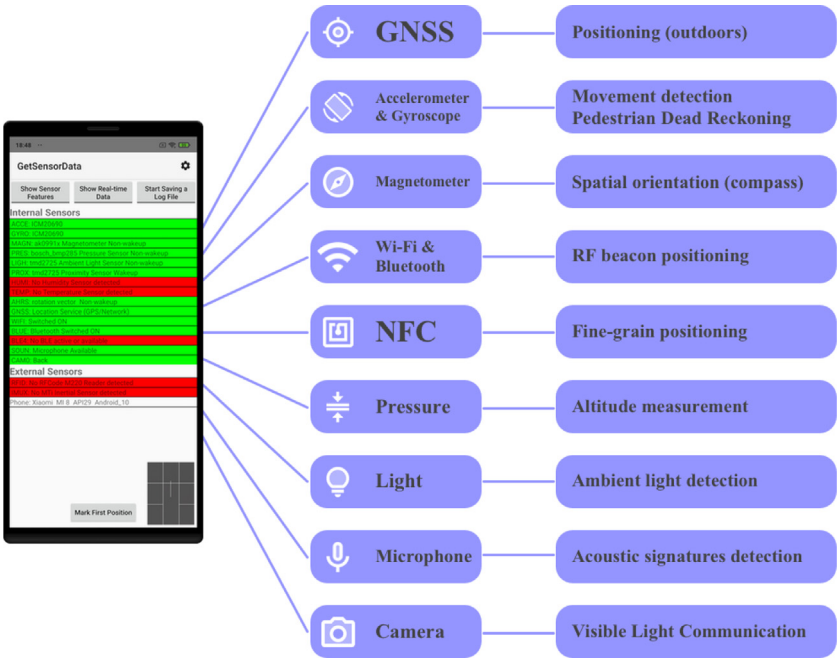


Fig. 2. Different smartphone sensors and possible application of the data acquired from them.

The rest of this article is organized as follows. In Section 2, the application architecture and functionality are introduced. A demonstration of the software capabilities is presented in Section 3. Section 4 highlights the benefits of using and extending the application. Finally, in Section 5, conclusions and future works derived from the application are extracted. Additionally, commented code snippets can be found in Appendix A.

2. Software description

GetSensorData is an Android application capable of showing data from the different sensors available in a smartphone on its main screen, as depicted in Fig. 2. Besides, data acquired from the sensors can also be permanently stored in a file using a standard format. Marks with timestamps can be included in the resulting files to ease synchronization tasks. The application capabilities can be expanded with new built-in or peripheral sensors when available. Other applications can take advantage of the acquired data for tasks such as indoor positioning Location-based Services (LBSs), Pedestrian Dead Reckoning (PDR) or Visible Light Communication (VLC), to mention a few.

GetSensorData is capable of retrieving data from hardware sensors, software sensors.[4] and external sensors connected to the smartphone via USB or Bluetooth. The software architecture allows for the inclusion of new sensors with minimal effort. A Radio Frequency Identification (RFID) reader[5] and an inertial motion unit sensor[6] are included in the current version of the application as examples. The application is compatible with Android 5.0 (Lollipop, API 21) and above, so it can run on approximately 98.6% of the existing devices[7]

2.1. Software architecture

GetSensorData uses a modular architecture at the core of which there is a hierarchy of classes that allows abstracting the sensors operation. The presentation is completely separated from the application logic, providing an easy-to-use standard interface applicable to all sensors, internal and external. Thanks to this interface, it is possible to share code and simplify the process of adding new sensors to the project. In Fig. 3, a diagram of the class hierarchy used in the application is shown, while Fig. 4 presents a flowchart of the class lifecycle within an Android activity. Methods on the left column are called directly by the programmer. Methods on the right are raised when a particular event happens so that the programmer can react accordingly.

The first step to include a sensor in the application is to create a subclass that inherits from `DataSensor`. Next, it is needed to complete methods with the appropriate code. Once the subclass is created, the operation is systematic, regardless of the sensor. When the activity is started, it is necessary to call the `connect()` method to connect the sensor. At the time the activity is destroyed, the `disconnect()` method must be called, so the resources assigned to the class copy are released. While the activity is running, the `startReading()` method allows reading data from the sensor. On the contrary, the `stopReading()` method stops the reading process. These methods are used according to the life cycle of the main activity.

The `DataSensor` class is accompanied by an interface called `DataSensorEventListener`. Using the delegation software design pattern [8], any class following this interface will be able to receive sensor events. As soon as the sensor is connected, the method `onDataSensorConnected()` is called. Conversely, when the sensor is disconnected `onDataSensorDisconnected()` is invoked. Finally, in case of having data to display `onDataSensorChanged()` is employed. Every time the sensor has data to display, it will notify the event, and the corresponding class instance will request the data to display them using the method `getStatus()`. This method receives the data destination as a parameter: screen or log file. Depending on the destination,

[4] https://developer.android.com/guide/topics/sensors/sensors_overview

[5] R&D Data Products RF Code M220 Mobile Reader: https://r-ddataproducts.com/

[6] Xsens MTi-G IMU: https://www.xsens.com/

[7] Android distribution dashboard: https://developer.android.com/about/dashboards

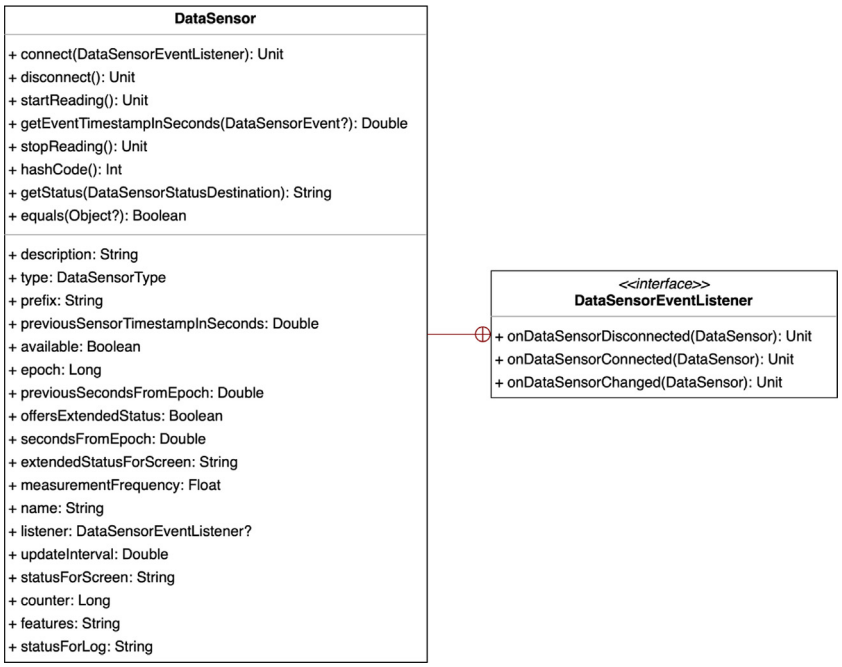


Fig. 3. DataSensor class hierarchy in the GetSensorData software.

Fig. 4. DataSensor class instance life cycle in the GetSensorData software.

the data format will be different. `DataSensor` also offers other sensor information about the sensor, such as its name, features or availability, to mention a few, though a series of attributes shown in Fig. 3. This design makes it possible to manage all sensors equally instead of operating different events separately, saving a lot of programming effort. Moreover, the application's main activity observes the events of each sensor and manages them equally, whether displaying their data in real-time or saving them in a file for further processing.

The main advantage of using this class hierarchy is that it eases the addition of new sensors to the application. Future programmers including new internal or external sensor do not have to worry about how to display the data: they only have to provide it. A commented example on how to create a class for sensor managing can be seen in Appendix.

2.2. Software functionalities

When the application is launched, it displays the interface screen shown in Fig. 5(a). At the top, the application name appears. The gear icon (⚙) at the right allows access to the application preferences panel, which can be seen in Fig. 5(c). Also there are four buttons that perform different actions, described below. In the center of the screen appears the list of the different sensors that GetSensorData can work with. Since not all Android smartphones have the same sensors, those present in the device show a green background, while the absent ones show a red background (see Fig. 5(a)). Therefore, it is straightforward to visualize the variety of sensors available in the smartphone. The "Internal Sensors" section shows the sensors provided by the smartphone, while the "External Sensors" section offers those connected to the smartphone using some communication protocol such as USB or Bluetooth. The camera is also treated as a regular sensor. A preview of the image acquired by the smartphone's camera is shown at the bottom right of the screen. Both the camera selected and the preview position within the main screen can be changed through the application preferences or disabled altogether.

The different action buttons on the main screen work as follows. The three ones at the top of the screen are toggle buttons. "Show Sensor Features" expands or collapses each row in the list of sensors, showing or hiding their features: manufacturer, model version, resolution, maximum range, power consumption and sampling rate (as shown in the gray areas of Fig. 5(b)). "Show Real-time Data" expands or collapses the real-time display for each of the available sensors (see the white areas in Fig. 5(b)). The sampling frequency displayed is estimated by the application and depends on the sensor's hardware and operating system, although it can be configured via the preferences panel shown in Fig. 5(c). "Start Saving a LogFile" starts a background task that saves all sensory data in a log file for later off-line analysis until the user touches it again to stop the application from saving data. The main screen keeps showing the sensory data to the user. The resulting file is stored in the folder "LogFiles_GetSensorData". Its filename is the unique combination of the date and time when the user started the recording process, and its format is described below in Section 3.1. The button "Mark First Position" is intended to insert reference lines through the log files. Although its original function was aiding with the definition of a GT, the references could mark the occurrence of any event. These marks are helpful to locate specific signals within the log file.

3. Illustrative examples

Although GetSensorData is a general-purpose application, and the sensory data acquired can be used for a wide variety of purposes, it was born as a tool to aid in positioning tasks, specifically during IPIN competitions. In preparation for the 2020 IPIN edition, the creators recorded a demonstration video to show how to use the application for data acquisition.[8]

Before gathering any sensory data, the first step consists of checking that all the required sensors are available. Those present are marked in green on the application's main screen, while the remaining ones are in red. The button "Show Sensor Features" is used to check the characteristics of each sensor needed, while "Show Real-time Data" is used to check if the application is registering the data collected by the sensors and the data sampling frequency.

When the user pushes the button "Start Saving a Log File" the data collection starts. The application shows a message to confirm the action, and the button shows the seconds elapsed since the data collection started. Before the user moves to the first position to be registered, it is advised to perform random movements for around 30 s and then remain static for another 30 s. The phone is moved in various orientations to remove the magnetometer biases in post-processing; then, it is kept in a static position to estimate and remove the gyro biases afterwards. This calibration step could be different depending on the nature of the sensors involved in the process. Once the sensors are calibrated, the button "Mark First Position" is touched to insert a "POSI" label in the log file. Then, the user walks at a constant speed from the first to the last test point, stopping in them to insert a "POSI" label. When the last test point is registered, the user remains static for 30 s and then repeats the process in reverse direction, from the last to the first point. After that, the user halts the data collection by touching the button "Stop Saving" to finish the process. Then, the log file can be shared with those interested in analyzing the data gathered. The data gathered during the process described in the video can be found at the aforementioned web address.

3.1. Log files

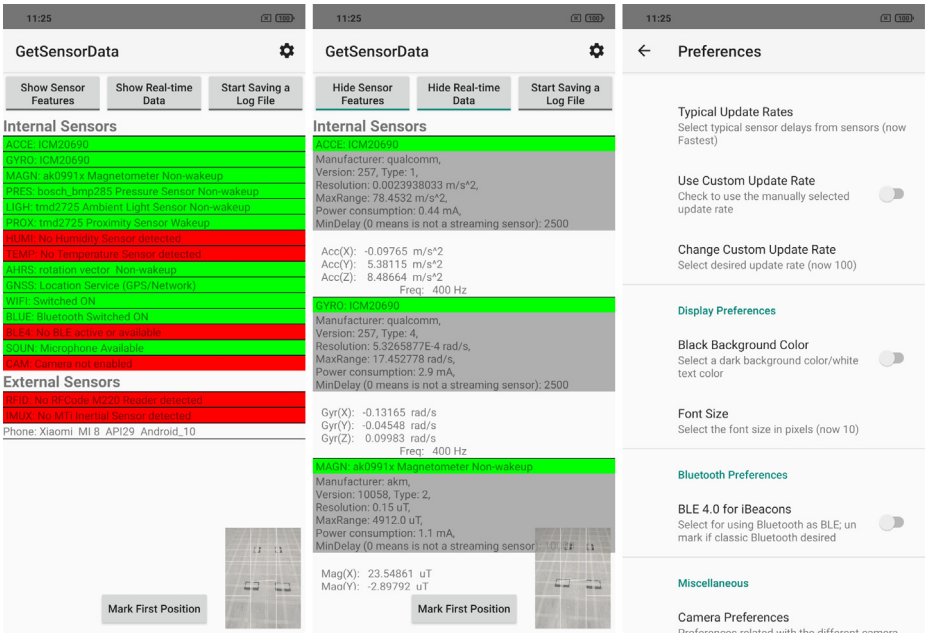
The log files created by GetSensorData are text files containing multiple rows from the continuous stream of sensory data generated by the smartphone. Each row is stored sequentially as new data are received, containing different values separated by semicolons. The row starts with a unique four-letter key (e.g. "WIFI", "ACCE", "MAGN", "BLE4", "GNSS") that identifies the sensor which the row is associated with. Due to the diversity of the data generated by each sensor, the values following the timestamp could include a different number of fields. An extract of an actual log file is presented in Listing 1. The entry in line 28 corresponds to a "POSI" mark that can be used to label GT locations, followed by timestamp.

4. Impact

4.1. LOPSI research group

GetSensorData was conceived as a tool to facilitate acquiring, saving and analyzing sensory data from a smartphone. GetSensorData has already been used as a local data collection tool in several works by co-authors of this article, although it has not been explicitly mentioned. In [9] it was used for location studies with standard luminaries using the internal light sensor

[8] http://indoorloc.uji.es/ipin2020track3/



(a) Main screen. (b) Sensor features and real-time data. (c) Settings.

Fig. 5. GetSensorData main screen appearance and preferences. Left: list of detected (green) and undetected (red) sensors for the smartphone; middle: instantaneous sensor readings; right: setup and preferences. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Listing 1: Example of a log file created with GetSensorData.

```
WIFI:21.644;835.679;test—CAR;00:0b:86:27:3e:82;2462;—87
WIFI;21.644;835.679;test—CAR;00:0b:86:27:35:12;2427;—90
WIFI;21.644;835.679;portal—csic;00:0b:86:27:3e:81;2462;—87
GYRO;21.697;835.705;—0.09682;—0.21533;—0.21197;3
ACCE;21.697;835.712;0.34177;2.88860;8.78074;3
MAGN;21.698;835.712;—1.74700;15.14200;—35.62900;3
AHRS;21.700;835.712;14.047199;—4.788099;4.554478;0.12372229;—0.03657122;0.03429850;3
GYRO;21.702;835.712;—0.09682;—0.21533;—0.21197;3
PRES;21.703;835.715;964.0800;3
LIGH;21.704;835.715;376.0;3
ACCE;21.707;835.720;0.41779;3.05919;8.76038;3
GYRO;21.707;835.720;—0.11270;—0.19945;—0.18937;3
AHRS:21.708;835.720:14.029488;—4.859563;4.467018:0.12356212;—0.03729048:0.03347217;3
LIGH;21.709;835.722;376.0;3
GYRO;21.709;835.725;—0.13225;—0.17135;—0.21441;3
ACCE;21.709;835.731;0.31484;3.03944;8.68257;3
MAGN;21.710;835.731;—2.21700;15.14200;—35.62900:3
AHRS;21.710;835.731;14.017381;—4.898848;3.918406;0.12328733;—0.03822555;0.02868964;3
GYRO;21.711;835.731;—0.13225;—0.17135;—0.21441;3
LIGH;21.711;835.734;376.0;3
GYRO;21.711;835.736,—0.15424;—0.14447,—0.27031;3
ACCE;21.712;835.740;0.21368;2.98078;8.72208;3
GYRO;21.712;835.740;—0.15424;—0.14447;—0.27031;3
AHRS:21.713;835.740;13.977528;—4.928859;3.829847;0.12292042;—0.03859377;0.02790756;3
LIGH;21.714;835.742;387.0:3
GYRO;21.714;835.745;—0.18265;—0.11973;—0.31887;3
ACCE;21.747;835.811;—0.33878;2.73298;10.35612;3
POSI;21.748;2;40.51296307;—3.34844;0;20
MAGN;21.748;835.811;—3.00200;15.47500;—35.62900;3
AHRS;21.748;835.811;13.456279;—4.567315;0.727160;0.11731420;—0.03882872;0.00162862;3
GYRO;21.749;835.811;—0.11240;—0.07239;—0.27855;3
LIGH;21.749;835.814;417.0;3
BLE4;21.750;Eddystone;ES:A3:78:5D:3E:9A;—94;201600000010;5954;17.0;20240;13591778
BLE4;21.751;iBeacon;C6:59:DE:CF:00:51;—73;—76;2016;3;b9407130—558—466e—aff9—25556b57fe6d
BLE4;21.751;iBeacon;F2:1E:E9:6C:5B:FB;—74;—76;2016;5;b9407f30—f5f8—466e—aff9—25556b57fe6d
BLE4;21.752;iBeacon;CO:F6:0F:77:8A:E9;—76;—76;2016;1;b940730—f5f8—466e—aff9—25556b57fe6d
```

and the external MTi-Xsens inertial sensor. Moreover, in [10] GetSensorData was used with external RFID readers from RFCode for tomography-based localization. Also, readings from external USB and Bluetooth sensors were first presented in these works.

4.2. IPIN competitions

Additionally, GetSensorData has been tested in the different editions of IPIN conference, starting with 2016 [2]. Specifically it was used in an off-site and offline positioning competition, providing beforehand the data gathering and positioning systems calibration. Multiple sources of information were supplied thanks to the log files created with GetSensorData. Those in charge of generating the log files moved continuously while recording data, walking and realistically holding the smartphone through four different buildings. Five teams took part in the competition, working on the same dataset with different approaches.

In [5] authors describe the results of the IPIN 2017 competition and compare them with other competition-based approaches (Microsoft [11], and Perf-loc [12]) and online evaluation websites, focusing on the smartphone-based (off-site) track. All the sensitive information, such as Wi-Fi Service Set Identifier (SSID)'s and Media Access Control (MAC) addresses were anonymized for the IPIN 2018 smartphone-based positioning challenge [13]. Each subsequent IPIN edition introduced new challenges such as the highest update rate (2019 [14] /2020 [15]), large open areas (2020 [15]), custom Bluetooth Low Energy (BLE) network (2021) or floor transition in an auditorium (2021).

GetSensorData was also used as a teaching tool [6] in a PDR tutorial. During the practical session, the attendees were requested to perform experiments after installing GetSensorData on their devices. Then, each of them could gather data and process it to understand and practice the PDR concepts and methods proposed in the tutorial.

4.3. Other research groups

After being presented in IPIN competitions, many researchers have taken advantage of GetSensorData features. A vision-aided positioning system was evaluated on the 4th floor of Sir Peter Mansfield Building at the University of Nottingham in Ningbo, China, using a camera and the PDR data collected with GetSensorData and a Huawei MT7-TL00 [16]. A calibration-free based on crowdsourced smartphone data was proposed in [17]. To validate a systematic approach to generating 3D path loss heat maps, the authors of [18] used GetSensorData to collect the reference data at each cubic meter in the hallways of two floors in the Arfa Karim block at the University of Gujrat. The authors of [19] proposed a method to detect behavior changes in the daily routines of people using BLE-based positioning information. In [20] the authors proposed a site survey approach that can automatically build maps of opportunistic signals for indoor localization by a dedicated surveyor using a smartphone. Authors of [21] proposed a transitional unsupervised learning algorithm able to segment massive unlabeled sequences and learn the relationship between two consecutive signal states and one-step motion. Finally, GetSensorData is also a part of the research work related to the magnetic field sensing for pedestrian and robot indoor positioning developed in [7].

4.4. Intended users

Although GetSensorData functionality in the previous examples remains practically the same, its architecture has been optimized. The modular and extensible class hierarchy presented in Section 2.1 is the main enhancement of the version presented in this paper. Also, the camera was recently included as a sensor so that GetSensorData could aid in the research of visible light positioning systems for smartphones developed in [22,23]. This work is part of the first author's PhD dissertation, scheduled to be published in the coming months.

From the user's perspective, Android applications follow an event-driven architecture, responding to user interaction whenever they attempt to perform an action. GetSensorData has been conceived with two types of users in mind: the end-user, who uses the application to obtain sensory data, querying it in real-time or processing it later, and the researcher who wishes to extend the functionality of the application, including new appropriate sensors for their work. To meet the end-user needs, it is enough to offer an application that provides the functionalities described in Section 2.2. However, satisfying the researchers needs is somewhat more complicated: adding new sensors to the application requires some programming skills, which is not guaranteed given that researchers interested in obtaining sensory data from a smartphone are part of a very heterogeneous group of people with very diverse programming skills. This software is an open tool devoted to data collection for smartphone-based applications, which will enable the community to provide results with a higher degree of versatility.

5. Conclusions

We have introduced GetSensorData, an open-source application for presenting, collecting and analyzing sensory data from Android smartphones. The development of this application has arisen from the collaboration of several prestigious research institutions, given their shared need to obtain sensory data from cell phones. The sensors can be embedded in the device or connected externally. The camera is also considered a sensor, and a preview can be shown in the application screen. This application is compatible with Android 5.0 (Lollipop, API 21) and above, so it can run on approximately 98.6% of the existing devices. GetSensorData software architecture allows for expandability with minimal effort. After more than six years of testing throughout different contexts, and many researchers taking advantage of the application, the application is ready to be open-source and available through a Git public repository that fosters collaboration via pull requests. Different repositories with documentation and complementary tools accompany the application. We hope GetSensorData keeps helping the community and growing as new sensors are included in modern smartphones, and external sensors of different nature are available.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This work has been supported by the Spanish Government and the European Regional Development Fund (ERDF) through Project MICROCEBUS under Grant RTI2018-095168-B-C54/C55, and by the Regional Government of Extremadura and ERDF-ESF under Project GR21054.

Listing 2: Ambient temperature DataSensor subclass.

```
class AmbientTemperatureDataSensor(
  context: Context, updateInterval: Double
): DataSensor(
  context, DataSensorType.AmbientTemperature, updateInterval
) {

  override fun getName(): String =
    if (sensor != null) {
      sensor.name
    } else {
      context.getString(R.string.ambient_temperature_sensor_not_detected)
    }

  override fun getFeatures(): String =
    if (sensor != null) {
      // Return sensor features
    } else {
      context.getString(R.string.no_features)
    }

  override fun getStatusForScreen(): String {
    // Return sensor status formatted to show directly to the user
  }

  override fun getStatusForLog(): String {
    // Return sensor status formatted to save in the log file
  }
}
```

Listing 3: DataSensor life cycle inside the activity.

```
public class MainActivity implements DataSensorEventListener {
  ...
  AmbientTemperatureDataSensor ambientTemperatureDataSensor;
  ...
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...
    ambientTemperatureDataSensor = new AmbientTemperatureDataSensor();
    ambientTemperatureDataSensor.connect();
    ...
  }

  @Override
  protected void onResume() {
    ...
    if (ambientTemperatureDataSensor.isAvailable()) {
      ambientTemperatureDataSensor.startReading();
    }
    ...
  }

  @Override
  protected void onPause() {
    ...
    ambientTemperatureDataSensor.stopReading();
    ...
  }

  @Override
  protected void onDestroy() {
    ...
    ambientTemperatureDataSensor.disconnect();
    ...
  }
  ...
}
```

Appendix A. Sample code snippets analysis

Listing 2 shows an abridged version of the Kotlin subclass in charge of collecting features and data from the ambient temperature sensor. The full code of the class can be found at https://bit.ly/3tLD3p1. It matches the data flow depicted in Fig. 4. Any researcher interested in including a new sensor in GetSensorData should follow the same pattern.

Besides, Listing 3 provides an instance of how the ambient temperature sensor is used inside the application main activity, programmed in Java. Only the code related to that class is shown, using an ellipsis to mark the omitted parts. The full code of the class can be found at https://bit.ly/3b9mWuX.

To finish, Listing 4 contains a reduced version of the Java code used to collect and display sensors data via a `DataSensorRecyclerView` instance. The full code can be found at https://bit.ly/3xDZZI3. This class exhibits the same behavior of a standard

Listing 4: DataSensor data is displayed in the screen.

```
@Override
public void onDataSensorChanged(@NotNull DataSensor dataSensor) {
  ...
  String status = dataSensor.getStatus(
    DataSensorStatusDestination.Screen
  );

  if (dataSensorsRecyclerView.getShowSensorRealTimeData()) {
    dataSensor.setStatusForScreen(status);
    dataSensorsRecyclerView.updateStatus(dataSensor);
  }
  ...
}
```

`RecyclerView`,[9] with some modifications to ease working with it in this project. Data is stored in a log file following a similar procedure.

Appendix B. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.softx.2022.101186.

References

[1] Ehrlich CR, Blankenbach J. Pedestrian localisation inside buildings based on multi-sensor smartphones. In: 2018 Ubiquitous positioning, indoor navigation and location-based services. 2018, p. 1–10.

[2] Torres-Sospedra J, Jiménez AR, Knauth S, Moreira A, Beer Y, et al. The smartphone-based offline indoor location competition at IPIN 2016: analysis and future work. Sensors 2017;17(3). http://dx.doi.org/10.3390/s17030557, URL https://www.mdpi.com/1424-8220/17/3/557.

[3] Laoudias C, Constantinou G, Constantinides M, Nicolaou S, Zeinalipour-Yazti D, et al. The airplace indoor positioning platform for android smartphones. In: 2012 IEEE 13th international conference on mobile data management. 2012, p. 312–5.

[4] Georgiou K, Constambeys T, Laoudias C, Petrou L, Chatzimilioudis G, et al. Anyplace: A crowdsourced indoor information service. In: 2015 16th IEEE international conference on mobile data management. Vol. 1. 2015, p. 291–4.

[5] Torres-Sospedra J, Jiménez AR, Moreira A, Lungenstrass T, Lu W-C, et al. Off-line evaluation of mobile-centric indoor positioning systems: the experiences from the 2017 IPIN competition. Sensors 2018;18(2). http://dx.doi.org/10.3390/s18020487, URL https://www.mdpi.com/1424-8220/18/2/487.

[6] Jiménez AR, Seco F, Torres-Sospedra J. Tools for smartphone multi-sensor data registration and GT mapping for positioning applications. In: 2019 International conference on indoor positioning and indoor navigation. 2019, p. 1–8. http://dx.doi.org/10.1109/IPIN.2019.8911784.

[7] Antsfeld L, Chidlovskii B. Magnetic field sensing for pedestrian and robot indoor positioning. In: 2021 International conference on indoor positioning and indoor navigation. 2021, p. 1–8. http://dx.doi.org/10.1109/IPIN51156.2021.9662599.

[8] Gamma E, Helm R, Johnson R, Vlissides J. Design patterns: elements of reusable object-oriented software. Addison Wesley; 1994.

[9] Jiménez AR, Zampella F, Seco F. Light-matching: A new signal of opportunity for pedestrian indoor navigation. In: International conference on indoor positioning and indoor navigation. IEEE; 2013, p. 1–10.

[10] Jiménez A, Seco F. Combining RSS-based trilateration methods with radio-tomographic imaging: Exploring the capabilities of long-range RFID systems. In: 2015 International conference on indoor positioning and indoor navigation. IEEE; 2015, p. 1–10.

[11] Lymberopoulos D, Liu J. The Microsoft indoor localization competition: Experiences and lessons learned. IEEE Signal Process Mag 2017;34(5):125–40. http://dx.doi.org/10.1109/MSP.2017.2713817.

[12] Indoor localization & tracking. 2021, URL https://perfloc.nist.gov/perfloc.php.

[13] Renaudin V, Ortiz M, Perul J, Torres-Sospedra J, Jiménez AR, et al. Evaluating indoor positioning systems in a shopping mall: The lessons learned from the IPIN 2018 competition. IEEE Access 2019;7:148594–628. http://dx.doi.org/10.1109/ACCESS.2019.2944389.

[14] Potortí F, Park S, Crivello A, Palumbo F, Girolami M, et al. The IPIN 2019 indoor localisation competition—Description and results. IEEE Access 2020;8:206674–718. http://dx.doi.org/10.1109/ACCESS.2020.3037221.

[15] Potortí F, Torres-Sospedra J, Quezada-Gaibor D, Jiménez AR, Seco F, et al. Off-line evaluation of indoor positioning systems in different scenarios: The experiences from IPIN 2020 competition. IEEE Sens J 2022;22(6):5011–54. http://dx.doi.org/10.1109/JSEN.2021.3083149.

[16] Yan J, He G, Basiri A, Hancock C. Vision-aided indoor pedestrian dead reckoning. In: 2018 IEEE international instrumentation and measurement technology conference. 2018, p. 1–6. http://dx.doi.org/10.1109/I2MTC.2018.8409599.

[17] Zhao Y, Zhang Z, Feng T, Wong W-C, Garg HK. GraphIPS: Calibration-free and map-free indoor positioning using smartphone crowdsourced data. IEEE Internet Things J 2021;8(1):393–406. http://dx.doi.org/10.1109/JIOT.2020.3004703.

[18] Haider A, Farooq MH, Mukhtar H, Ali MU. A systematic approach to generate 3D path loss heat maps for WIFI indoor positioning. Eng Proc 2021;12(1). http://dx.doi.org/10.3390/engproc2021012106, URL https://www.mdpi.com/2673-4591/12/1/106.

[19] Martín AJ, Gordo IM, Gómez DG, de Villa SG, Plaza SL, et al. BLE-based approach for detecting daily routine changes. In: 2021 IEEE international symposium on medical measurements and applications. 2021, p. 1–6. http://dx.doi.org/10.1109/MeMeA52024.2021.9478752.

[20] Liang Q, Liu M. An automatic site survey approach for indoor localization using a smartphone. IEEE Trans Autom Sci Eng 2020;17(1):191–206. http://dx.doi.org/10.1109/TASE.2019.2918030.

[21] Ye X, Huang S, Wang Y, Chen W, Li D. Unsupervised localization by learning transition model. Proc ACM Interact Mob Wearable Ubiquitous Technol 2019;3(2). http://dx.doi.org/10.1145/3328936.

[22] Gutiérrez JD, Álvarez FJ, Aguilera T, Paredes JA, Morera J. Visible light positioning for smartphones based on biphase mark coding: a proof of concept. In: 2018 International conference on indoor positioning and indoor navigation. 2018, p. 4.

[23] Gutierrez JD, Aguilera T, Álvarez Franco FJ, Aranda FJ. A blender-based simulation tool for visible light positioning with portable devices. In: 2022 International instrumentation and measurement technology conference. 2022, p. 6.

[9] https://developer.android.com/guide/topics/ui/layout/recyclerview