2021 International Conference on Indoor Positioning and Indoor Navigation (IPIN), 29 Nov. – 2 Dec. 2021, Lloret de Mar, Spain

# Indoor Positioning Using the OpenHPS Framework

1st Maxim Van de Wynckel
*Web & Information Systems Engineering Lab*
*Vrije Universiteit Brussel*
1050 Brussels, Belgium
mvdewync@vub.be

2nd Beat Signer
*Web & Information Systems Engineering Lab*
*Vrije Universiteit Brussel*
1050 Brussels, Belgium
bsigner@vub.be

***Abstract*—Hybrid positioning frameworks use various sensors and algorithms to enhance positioning through different types of fusion. The optimisation of the fusion process requires the testing of different algorithm parameters and optimal low- as well as high-level sensor fusion techniques. The presented OpenHPS open source hybrid positioning system is a modular framework managing individual nodes in a process network, which can be configured to support concrete positioning use cases or to adapt to specific technologies. This modularity allows developers to rapidly develop and optimise their positioning system while still providing them the flexibility to add their own algorithms. In this paper we discuss how a process network developed with OpenHPS can be used to realise a customisable indoor positioning solution with an offline and online stage, and how it can be adapted for high accuracy or low latency. For the demonstration and validation of our indoor positioning solution, we further compiled a publicly available dataset containing data from WLAN access points, BLE beacons as well as several trajectories that include IMU data.**

***Index Terms*—hybrid positioning, indoor positioning, process network, fingerprinting, pedestrian dead reckoning**

## I. Introduction

Knowing the real-time position of a person or object is a common problem that can be solved via a broad range of technologies and algorithms. Existing modular positioning frameworks offer *location providers* to handle the positioning for a given technology, but they are often not flexible in adapting the flow and low-level fusion of information in order to fulfil specific requirements. For indoor positioning systems, the use cases can range from asset tracking in large warehouses, location context in implicit human-computer interaction, to the use of a positioning system for company-wide contact tracing during an epidemic [1]. Depending on the requirements for the sampled position, different positioning and fusion techniques might offer the best results.

Our OpenHPS system [2] aims to address the existing lack of control by providing a modular low-level positioning framework that can easily be modified depending on the requirements and available technologies. OpenHPS is designed to support a variety of implementations, ranging from indoor positioning to object tracking on a smaller scale such as a game board. These implementations are realised by using a process network design for controlling how sensor data is managed. Each node in the process network represents an operation in the flow of sensor information, ranging from the processing of data to controlling how the data is fused for generating an output position. Despite its low-level process network design, OpenHPS has been developed with the goal of position processing to support developers who want to develop a customised positioning solution. It solves common issues such as transformations, low- and high-level sensor fusion, data storage and offers common positioning algorithms.

978-1-6654-0402-0/21/$31.00 © 2021 IEEE

In this paper, we explore and demonstrate the use of OpenHPS for an indoor positioning use case based on commonly used positioning methods for Wi-Fi signals, Bluetooth beacons and pedestrian dead reckoning. Our main contribution is the introduction of the flexible OpenHPS framework and its use for indoor positioning, with several modular components that can be utilised for this particular use case. We showcase and validate the framework based on a new indoor positioning dataset containing a data collection of 220 anonymised WLAN access points (including SSID grouping and broadcasting frequency), 11 BLE beacons with a known position and IMU data (i.e. orientation, acceleration, rotation rate and magnetometer data) recorded at 140 reference positions in four directions. In addition to this dataset, we provide several trajectories and a GeoJSON feature set with polygonal areas of interest.

## II. Related Work

In the research field of indoor positioning and navigation, a broad range of hybrid positioning frameworks exist. Hightower et al. [3] presented the seven layer location stack based on five design principles. These principles offer a good baseline for the requirements of an extensible hybrid positioning system. MiddleWhere [4] is such a framework, using a similar layered structure of sensor data, context reasoning and semantic activities through a location-based service. It allows developers to focus on positioning within a reference space, such as a room, without the need to work with geometric coordinates. MiddleWhere uses the concept of symbolic locations to define region- and object-based locations. Positioning technologies are added through adapters, but only provide an interface to the system as a completed positioning method. The fusion of these technologies is done by a fixed probabilistic reasoning of the determined symbolic locations.

Most positioning systems have a specific design goal. LearnLoc [5] is a hybrid system that aims for power-efficient indoor positioning rather than obtaining the most accurate

position. Bekkelien and Deriaz [6] introduced the Global Positioning Module (GPM) framework for in- and outdoor positioning. GPM provides a uniform interface to different position providers which are fused in a kernel that selects the position based on criteria such as probability, accuracy and precision. The position providers and kernels are implemented on a high level of abstraction, allowing no room for developers to choose different algorithms or fusion techniques. However, their methodology of using different criteria to determine the fusion offers some flexibility to implement a system for a particular use case.

Similarly, Ficco and Russo [7] presented a technology-independent hybrid positioning middleware called HyLocSys that also accepts some position criteria. Position estimators, representing different technologies, provide positions when a user requests their current position. Sensor fusion combines these estimated positions into a final response. With the middleware being an extension of the JSR-179 [8] specification, these pull requests accept criteria such as the preferred response time and expected accuracy. In addition to the geographical positions offered by most frameworks, HyLocSys provides geometric, symbolic and hybrid location models. Symbolic locations represent abstract places such as buildings, floors and rooms that are positioned relatively to each other.

While not being considered a positioning framework, the IndoorLoc Platform [9] offers a web-based user interface for evaluating different positioning algorithms with configurable parameters. The offered flexibility is limited to fixed datasets and k-NN or probabilistic fingerprinting, but shows the need for experimenting with and configuring a positioning system.

One of the more widely used systems is the Robotics Operating System (ROS) [10]. While ROS handles many different robotics aspects other than positioning, it provides relevant features such as multi-sensor fusion for indoor positioning and transform frames [11] that can be used to transform sensor data to the reference coordinate system used by the robot.

Last but not least, IndoorAtlas [12] is a well established Platform as a Service (PaaS) solution for combining Wi-Fi, GPS, Bluetooth beacons, dead reckoning and geomagnetic positioning. The focus is on allowing end users to author and configure the system for an indoor environment.

Our proposed OpenHPS positioning system should adhere and support specifications such as WGS 84 [13] when working with geographical positions. However, unlike most of the related work discussed in this section, we also want to support use cases with non-geometric coordinates. On the other hand, the work by Ficco and Russo [7] offers a good type of hybrid location, but it is still heavily focused on geometric positions. With our framework we want to implement symbolic locations on a level that still allows non-geometric positions.

Positioning methods and algorithms are often represented under the term *providers* that are optionally combined via high-level decision fusion [14]. In our framework, we want to separate providers into generic algorithms and positioning methods that can easily be interchanged. This does not only improve the extensibility, but also enables low-level sensor fusion. Similar to the *Indoor Location*[1] framework offering Android and iOS position providers, we provide all components as open source. However, with our OpenHPS framework we do not want to limit ourselves to smartphone applications and black box position providers.

The Geolocation API [15], JSR-179 and HyLocSys allow for the specification of accuracy or some other criteria when requesting a position. Unlike high-level APIs that hide the underlying technologies, OpenHPS addresses developers with an understanding of the available hardware and positioning techniques that influence these criteria. Additional layers of abstraction can simplify the configuration, but should still allow developers to optimise the algorithm parameters.

The persistence of landmarks, as realised in JSR-179, is an important requirement that is extended to fingerprinting information and cached position storage in OpenHPS. This cache allows positioning algorithms to use historical information.

## III. OpenHPS Framework

OpenHPS[2] is an open source hybrid positioning framework implemented in TypeScript. The system is split into individual modules that provide extra functionality on top of a core component. The core component of the OpenHPS framework is a process network designed to sample sensor data to a position while other components extend this core functionality with different data storage types, positioning techniques, abstractions and communication nodes.

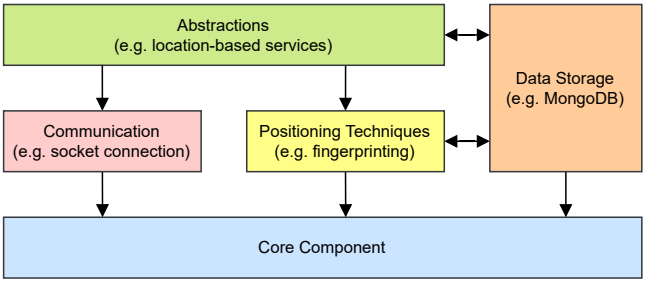


Fig. 1: Component architecture of OpenHPS

A general overview of the main categories of components provided by OpenHPS is given in Fig. 1. The core component can be extended with additional communication nodes that handle data transfers from hardware or to remote parts of the process network. Positioning algorithms such as fingerprinting techniques or SLAM might expand on the basic data types and processing nodes provided by the core API. Additional data storage options such as MongoDB or MobilityDB [16] can persist these data types, trajectories or custom data generated by parts of the process network. This custom data might range from processed fingerprints to historical information from signal filter nodes. Finally, components can abstract parts of the underlying process network or data storage. An example of such an abstraction is a location-based service offering a high-level API for the current position of an object.

[1]https://www.indoorlocation.io
[2]https://openhps.org

The core OpenHPS API requires all data types that are handled by the process network to be serialisable. This enables us to scale and decentralise parts of the network to multiple (remote) processes or (web) workers, bypassing the single-threaded limitation of JavaScript.

### A. Design Principles

OpenHPS has been designed around four main actors that can be identified in most positioning systems:

- **Tracked actor**: This is an actor that can be tracked during the online positioning stage. In indoor positioning, it might represent the user or object that is being tracked in an environment. Data that is processed in the network should have a reliable type and content. We process `DataObjects` encapsulated in `DataFrames` (see Section III-C), providing a defined scope on how generic parts of our network should handle information.
- **Tracking actor**: This actor is responsible for tracking a tracked actor. It can be identical to the tracked actor (e.g. a user's smartphone) or some separate technology such as a camera tracking the movement of a user. This actor has the highest priority and slow consumers or computing actors must not result in outdated sensor information. Rather, developers should be given the opportunity to control what happens with any potential overflow of information that cannot be processed timely.
- **Calibration actor**: Some positioning methods require a calibration or setup before they can be used. Unlike the tracked actor, the purpose of a calibration actor is to train and calibrate how a tracking actor will be used during the system's online stage.
- **Computing actor**: The computing actor is responsible for providing the final position output. It combines the data generated by one or multiple tracking actors and processes the data by using specific positioning algorithms to providing the absolute position of tracked actors.

Note that our four actors can be mapped to layers and design principles in the location stack [3]. Furthermore, computing actors represent an important component of the framework, as they are responsible for the processing and fusion of sensor data from multiple sensors.

### B. Process Network

The process network consists of nodes that handle the creation, processing and storage of data frames. We identify three main types of nodes:

- **Source node**: A source generates data and normally represents a sensor generating data automatically or on request.
- **Processing node**: A processing node provides an abstraction on top of the push and pull functionality to simplify the processing of sensor data or data objects.
- **Sink node**: A sink node stores data objects upon receiving a data frame. Once saved, an event is sent upstream to indicate that the processing of the frame and its objects has been completed.

In addition to these three main nodes, the core component offers process shapes for controlling the flow and fusion of data. The component-based structure of the OpenHPS framework enables extensions and abstractions of the three main nodes to support additional sensor sources and processing techniques. Future components might offer nodes that perform context reasoning and fusion.

Our framework uses a push-pull-based stream for sampling sensor data based on existing stream-based software architectures. However, we optimised the framework for processing and handling positioning data. Source nodes that actively produce information, such as an IMU sensor, can push information. Pull request actions trigger a *push* when a node is able to respond to the request. This behaviour is similar to Akka Streams [17], but unlike reactive streams our framework does not use the behaviour to implement back pressure in the system. Both the push and pull requests can be executed asynchronously, similar to reactive streams [18].

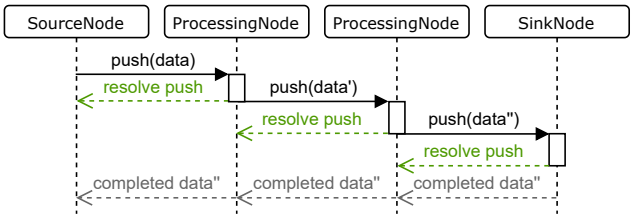


Fig. 2: Pushing data in the process network

Fig. 2 shows data being pushed by a source node and handled by two processing nodes. Once a downstream node is ready with the frame, it resolves the promise signalling to the upstream node that new data can be accepted. Sink nodes emit an event upstream, indicating that data has been persisted.

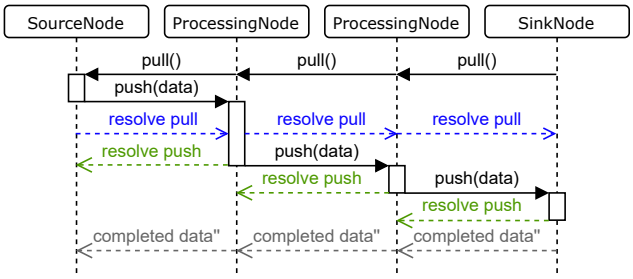


Fig. 3: Pulling data in the process network

Other than a push-based source, Fig. 3 illustrates a sink initiating a pull. This pull will be responded to by pushing new data, similar as shown earlier in Fig. 2. While we use the *pull* terminology, this action indicates a request to an upstream node to push new information if available and does not respond directly with the data. This allows us to process data frames sequentially, as required to perform optimal positioning.

### C. DataFrame and DataObject

A `DataObject` is the *tracking* or *tracked* actor in our system. These uniquely identified objects can contain a position and multiple relative positions to other objects or landmarks.

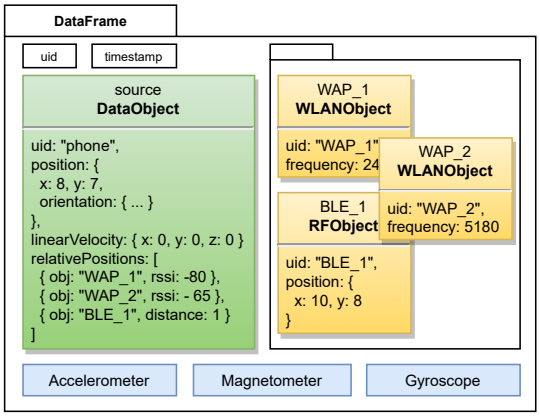


Fig. 4: Data frame containing objects and sensor data

Data objects are encapsulated in `DataFrames` generated by a source node. This timestamped frame contains data objects pertaining to the source that generated the frame. The structure of a fictional data frame generated by a smartphone application is outlined in Fig. 4. The source object (indicated in green) contains all previously persisted data such as the latest position and velocity of the sensor generating the data frame. In our illustrative example, we include three relative positions to other landmarks. The source node includes the data objects that are used inside these relative positions (data objects in yellow). Raw sensory data such as acceleration or images in a video stream are included in the data frame and can be accessed by processing nodes. Data frames and data objects can be extended to contain different data. A `WLANObject` named "WAP_1" is an extended data object containing information about the transmission frequency as shown in Fig. 4.

### D. Services

Each OpenHPS positioning model enables the use of services that can be accessed by nodes in the process network for data storage or the shared (pre)processing of information.

*1) Data Service:* A data service is a shared service for persisting information in the process network. Source nodes can access these services to load data objects and sink nodes to persist data objects. Data services can be compared to the storage capabilities of JSR-179 [8]. In addition to the persistence of data objects, nodes can store data about individual data objects. This allows filter nodes (e.g. an SMA or Kalman filter) to store historical information of a data object that is only relevant for one particular node in the process network.

*2) Fingerprinting Service:* In the *@openhps/fingerprinting*[3] component we provide a data object service for handling fingerprints. Other than persisting these fingerprints, the service can preprocess fingerprints by aggregating, interpolating or extracting features.

*3) Location-based Service (LBS):* A positioning model can be extended with a location-based service that uses the process network to offer a push- or pull-based LBS with similar high-level functionality such as the Geolocation API [15].

[3]https://github.com/OpenHPS/openhps-fingerprinting

```
1  const service = new LocationBasedService<
2    DataObject,
3    GeographicalPosition
4  >();
5
6  // Use the service in a positioning model
7  ModelBuilder.create()
8    .addService(service)
9    .from(/* ... */).via(/* ... */).build(/* ... */);
10
11 // Get the current position (cache or pull)
12 service.getCurrentPosition("myuser").then(pos => {
13   console.log(pos);
14 }, { maximumAge: 60000 });
15 // Watch position changes updated by the model
16 service.watchPosition("myuser", pos => {
17   console.log(pos);
18 }, { forceUpdate: false });
```

Listing 1: Location-based service

Listing 1 illustrates how a location-based service is added to the model (line 8). The current position of a certain object can be requested by using a simple API call of this service (lines 12–14). If the position is outdated, a pull is performed on the positioning model regardless of its construction or complexity.

### E. Position and Spaces

Similar to other frameworks [19], [20], we distinguish between *absolute* and *relative* positions. Absolute positions represent physical 2D, 3D or geographical positions within a certain area, while relative positions represent a position relative to another data object or landmark (e.g. relative distance, angle, velocity or signal strength). Each position might contain a timestamp on when the position was last updated, an orientation represented as a quaternion and both the linear and angular velocity.

Each absolute position is relative to a reference space, supporting the transformation of a position, orientation and velocity to a global reference space specified by the developer. Transformation spaces can be dynamically manipulated by other nodes in the process network, allowing processing nodes to control or calibrate how sensor data should be transformed. This behaviour is similar to the transform package of ROS [11], where the data of sensors on moveable parts of a robot can be transformed to a single coordinate space.

Symbolic spaces represent a high-level API extension of reference spaces aimed for indoor environments. They add the following capabilities on top of reference spaces:

- **Spatial hierarchy**: Spatial hierarchy is already supported in reference spaces using the parent identifier. However, symbolic spaces add boundaries that can be used to indicate whether an object is inside a space.
- **Graph connectivity:** The ability to connect spaces such as rooms, hallways, staircases and floors in order to support navigation applications or improve position estimation.

- **Geocoding:** Symbolic spaces can be converted to an absolute position in the global reference space. Similarly, an absolute position can be converted to the most likely symbolic space through reverse geocoding.
- **GeoJSON export:** Spaces might be exported to GeoJSON [21] features, aiding the storage and query capabilities in MongoDB or other data services.

Symbolic spaces can be treated as *symbolic locations* [4], [7], [22] with the ability to be converted to absolute positions.

```
const building = new Building("PL9")
  .setBounds({
    topLeft: new GeographicalPosition(50.8203, 4.3922),
    width: 46.275, height: 37.27, rotation: -34.04
  });
const floor = new Floor("PL9.3")
  .setBuilding(building)
  .setFloorNumber(3);
const office = new Room("PL9.3.58")
  .setFloor(floor)
  .setBounds([
    new Absolute2DPosition(4.75, 31.25),
    new Absolute2DPosition(8.35, 37.02),
  ]);

const object = new DataObject("myuser");
// Set the position relative to the floor space
object.setPosition(
  new Absolute2DPosition(6.55, 34.135, LengthUnit.METER)
), floor);
// Get the position relative to the global reference space
office.getPosition(); // (lat: 50.8204, lng: 4.3922)
// Get the position relative to the floor
office.getPosition(floor); // (6.55, 34.135)
```

Listing 2: Symbolic space creation and usage

Listing 2 shows the creation and usage of three symbolic spaces used in the evaluation of our indoor positioning demonstration presented later in Section IV-B. Boundaries can be specified through different methods. In this example, the boundaries of a building are defined via the top left corner, the width and height of the building in metres as well as the building's orientation (angle) relative to grid north. For the boundaries of the floor and a specific office room, we provided two boundary points that create a rectangular symbolic space. Alternatively, polygonal shapes can be used to define the boundaries.

While we have presented the core components and main features of the OpenHPS framework, additional technical details and examples can be found in [2].

## IV. Indoor Positioning Demonstration

In order to demonstrate the use of OpenHPS for indoor positioning, we created a process network with a server, two Android applications and a socket connection for transmitting sensor data to a server and feedback back from the server.

The server is implemented based on node.js[4] and handles the storage of fingerprints and position processing, while the two applications have been developed using React Native[5]. In our *@openhps/react-native*[6] component we provide several source nodes for interfacing with native sensors.

The complete positioning model is shown in Fig. 5. Two socket source nodes on the server (indicated in green) handle the server endpoints for the offline and online stage application. In the offline stage, features from objects within data frames are stored as fingerprints.

The fingerprint service used to store fingerprints is shared with the online stage. For the scope of our evaluation, we used various positioning methods ranging from BLE multilateration and cell identification using 11 BLE beacons, WLAN fingerprinting and BLE fingerprinting. A high-level position fusion node fuses the positions based on their accuracy [14]. Finally, the calculated position is sent back to the mobile application through the socket sink node (orange) as indicated in Fig. 5. The effectiveness of OpenHPS as a hybrid positioning solution is validated with two scenarios in Sections IV-B and IV-C.

```
GraphBuilder.create()
  .from(new IMUSourceNode({
    source: new DataObject(phoneUID),
    interval: 20, // 50 Hz (20ms interval)
    sensors: [
      SensorType.ACCELEROMETER, SensorType.ORIENTATION
    ]
  }))
  .via(new SMAFilterNode(
    frame => [frame, "acceleration"], { taps: 10 }
  ))
  .via(new GravityProcessingNode({
    method: GravityProcessingMethod.ABSOLUTE_ORIENTATION
  }))
  .via(new PedometerProcessingNode({
    minConsecutiveSteps: 1,  stepSize: 0.40
  }))
  .to("pedometer-output")
```

Listing 3: Graph shape for pedestrian dead reckoning

In the online stage we use the acceleration and orientation from an IMU source node to perform pedestrian dead reckoning as illustrated in Listing 3. A simple moving average filter on the acceleration will smooth the raw sensor data. Next, a gravity processing node uses the orientation and acceleration to extract the gravity and linear acceleration. Finally, we use a windowed average peak counting algorithm to perform step detection. The dead reckoning is fused with the feedback from the server—indicated by the purple fusion node of the online-stage application in Fig. 5—after which it is shown to the user and stored via a data object service. This finalised position is fed back to the position fusion node with a delay of 150ms and the velocity applied. Feedback from the server might contain a delayed position due to the time it takes to complete a scan and process it on the server and therefore we apply the last known velocity to the position returned from the server.

[4]https://nodejs.org/en/

[5]https://reactnative.dev

[6]https://github.com/OpenHPS/openhps-react-native

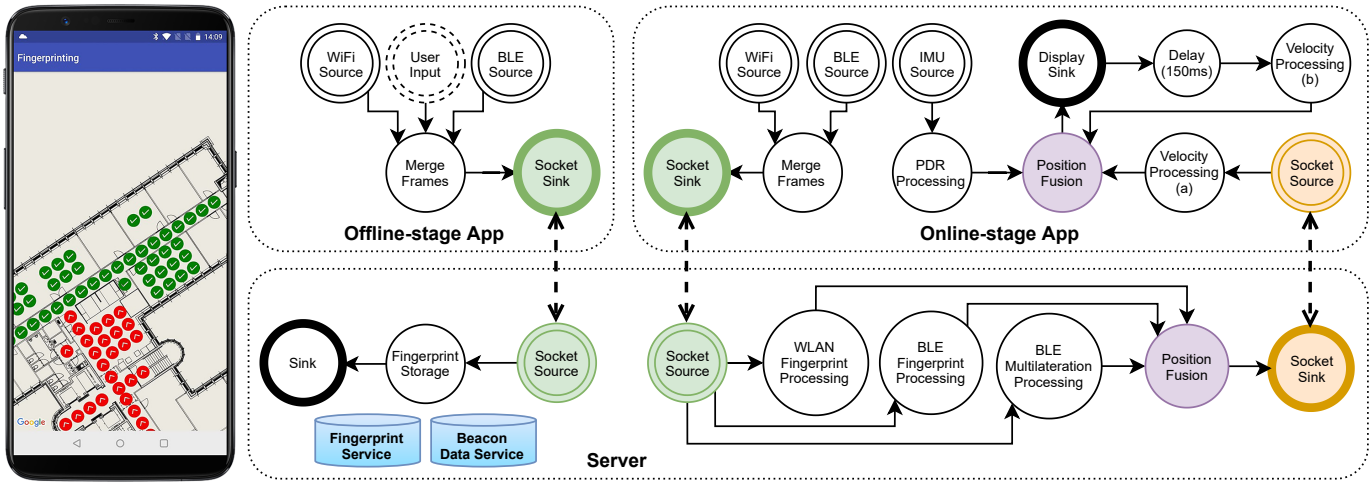


Fig. 5: Positioning model for server, offline and online application

### A. Dataset

For the evaluation or our positioning model, we created a fingerprinting dataset of a single floor in the building of our research lab [23]. A visual representation of our dataset is shown in Fig. 6. The dataset was recorded with a calibration application collecting information from WLAN access points, BLE beacons with a known position (blue) and an IMU sensor.

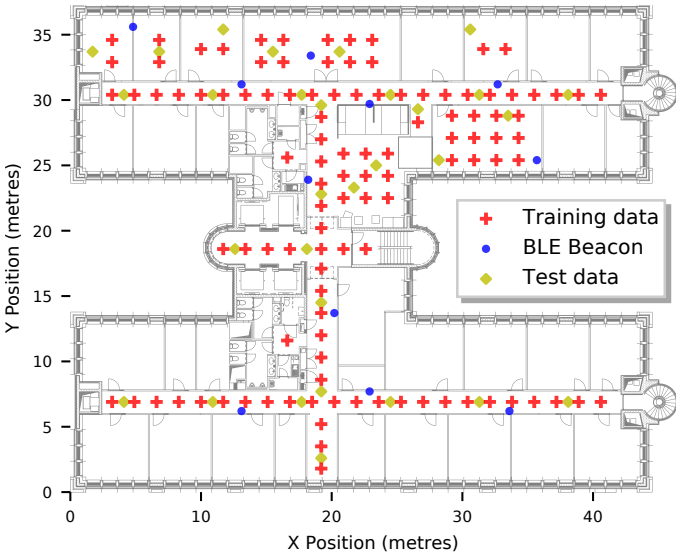


Fig. 6: Fingerprinting dataset data points

Each of the 110 training data points (red) and 30 test data points (green) have been collected in four directions (up, right, bottom and left) by standing still for 20 seconds with a phone held at chest height. In our dataset, we provide the raw data of all WLAN scans, BLE advertisements as well as IMU data including device orientation, acceleration, rotation rate and magnetometer data. Each detected access point is anonymised, but we provide information about SSID groups and the frequency of the access point. This high-dimensional information allows developers to use the dataset to experiment with different fingerprinting techniques that can potentially take the orientation and signal propagation into account for different Wi-Fi broadcasting frequencies [24].

We use the symbolic space abstraction that has been introduced in Section III-E to create symbolic spaces for the rooms, corridors two lobbies and toilets. These symbolic spaces will be used to determine the hit rate and are illustrated in Fig. 7 exported as GeoJSON polygonal features.


Fig. 7: Symbolic spaces in GeoJSON format

### B. Test Data Points

Our first evaluation of stationary test data points uses WLAN and BLE information to determine the position and symbolic location. For this test we used aggregated RSSI (received signal strength indicator) results.

We configured the WLAN fingerprinting on our server positioning model shown in Listing 4. On lines 2–5, we configure the preprocessing fingerprinting service. The flexibility

```
1  ModelBuilder.create()
2    .addService(new FingerprintService(
3      new MemoryDataService(Fingerprint), {
4      classifier: "wlan", defaultValue: -95
5    }))
6    .addShape(GraphBuilder.create() // ONLINE MODE
7      .from(/* ... */)
8      .via(new KNNFingerprintingNode({
9        weighted: true,  k: 4, classifier: "wlan",
10       weightFunction: WeightFunction.SQUARE,
11       similarityFunction: DistanceFunction.EUCLIDEAN
12     }))
13     .to(/* ... */))
14   .addShape(GraphBuilder.create() // OFFLINE MODE
15     .from(/* ... */)
16     .via(new FingerprintingNode({
17       classifier: "wlan"
18     }))
19     .to(/* ... */))
20   .build();
```

Listing 4: Positioning with fingerprinting parameters

of our system allows developers to choose how fingerprints are stored, normalised and aggregated. If needed, this service can be replaced with a custom pre-processing algorithm. Lines 16–18 show the creation of an offline fingerprinting node. This is an object processing node extracting features of objects and storing them in the fingerprinting service. The online stage (lines 8–12) can use processed fingerprints to obtain a position. In this particular test, we used a weighted k-NN algorithm that is configured similarly to the parameters used by RTLS@UM in the EvAAL competition of 2015 [25]. The use of general parametrised processing nodes offers great flexibility for developers to tweak the positioning system. In this evaluation we removed the pedestrian dead reckoning and feedback loop from our positioning model.

Table I shows the average, minimum and maximum error for different positioning techniques, along with the standard deviation and failed points. Failed points indicate test reference points for which the positioning technique was not capable to determine a position. In the case of BLE positioning, these are the positions where not enough BLE beacons were in range. The symbolic space hit rate represents the amount of test data points that were assigned to the correct symbolic space.

The modularity of our framework allows developers to rapidly adapt and test the sensor fusion for a specific use case. Other than trying to determine the most accurate average error, the goal of a positioning system might be to get the most accurate hit rate, increase the update frequency or minimise the energy consumption. In our chosen sensor fusion, we adapt the accuracy of a certain positioning technique based on the available information (e.g. BLE beacons in range) to achieve a higher symbolic space hit rate.

### C. Trajectories

In addition to stationary data points, we included several trajectories in our dataset that include IMU data on top of the WLAN and BLE data. The process network in our online application sends the WLAN and BLE data to the server, where it is processed similarly to the test data points in Section IV-B, while the IMU data is used locally in the application to perform pedestrian dead reckoning. Trajectory sensor information was collected by keeping the phone at chest height while performing the trajectory at a normal walking pace. Other than the stationary points, the update frequency and accuracy is more important than the symbolic hit rate.

TABLE I: Average, minimum and maximum X/Y position error compared to the fused position

| Positioning Techniques | | |
|---|---|---|
| WLAN Fingerprinting (k=4) | **failed points** | 0 |
| | **average error** | 1.23 m |
| | **minimum error** | 0.01 m |
| | **maximum error** | 4.77 m |
| | **standard deviation** | 1.04 m |
| | **symbolic space hit rate** | 95.82% |
| BLE Fingerprinting (k=3) | **failed points** | 6 |
| | **average error** | 3.23 m |
| | **minimum error** | 0.17 m |
| | **maximum error** | 15.39 m |
| | **standard deviation** | 2.69 m |
| | **symbolic space hit rate** | 80.83% |
| BLE Multilateration | **failed points** | 12 |
| | **average error** | 4.92 m |
| | **minimum error** | 0.74 m |
| | **maximum error** | 19.26 m |
| | **standard deviation** | 3.50 m |
| | **symbolic space hit rate** | 52.50% |
| Sensor Fusion (WLAN + BLE) | **failed points** | 0 |
| | **average error** | 1.37 m |
| | **minimum error** | 0.01 m |
| | **maximum error** | 9.75 m |
| | **standard deviation** | 1.26 m |
| | **symbolic space hit rate** | 96.67% |

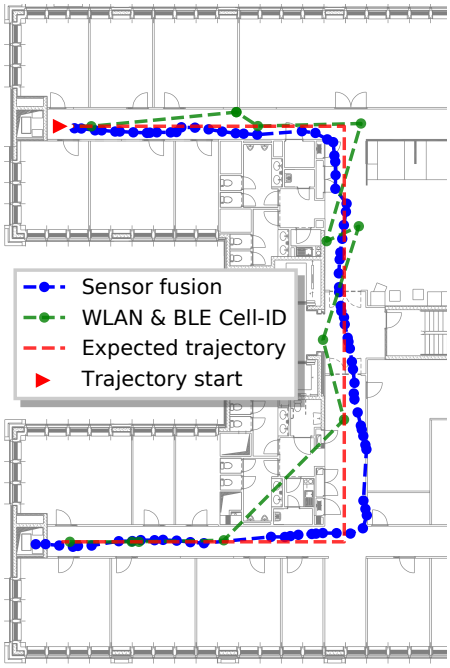


Fig. 8: Test trajectory with WLAN, BLE and IMU data

The expected trajectory is shown in red in Fig. 8. We determined the error by comparing the last known position with the actual expected position in the trajectory. While WLAN positioning and BLE cell identification can show a visual representation of the complete route, it only consists of 13 data points that are not synchronised with the user's real-time position. This delay is due to the scan duration and the processing time on the server.

TABLE II: Sensor fusion comparison for test trajectory

| **Positioning Techniques** | | |
|---|---|---|
| WLAN + BLE | **average error** | 3.29 m |
| | **maximum error** | 9.60 m |
| | **standard deviation** | 2.09 m |
| | **average update frequency** | 3.04 s |
| WLAN + BLE + IMU | **average error** | 1.26 m |
| | **maximum error** | 3.10 m |
| | **standard deviation** | 0.77 m |
| | **average update frequency** | 0.52 s |

In Table II we show the maximum and average error for our test trajectory with and without IMU data. The delay caused in the fingerprinting in combination with the slow update frequency causes a larger error compared to the real-time position during the trajectory. Note that flexibility of OpenHPS allows developers to experiment with different positioning algorithms and fusion techniques to further optimise the system.

## V. Conclusion and Future Work

We presented the OpenHPS framework and its use as an indoor positioning solution. We used a server with socket endpoints that stores fingerprints and fuses WLAN and BLE data to a predicted position that is transmitted back to the mobile application where it is fused with processed IMU data. We have illustrated the modularity with our symbolic spaces that build on top of the OpenHPS core component to provide symbolic locations, similar to HyLocSys [7]. Different from related hybrid positioning frameworks offering specific location providers, we demonstrated how nodes in the OpenHPS process network can be constructed and configured to support various technologies.

For our indoor positioning demonstration, we recorded a new open source dataset of a single floor in our research lab with help of the OpenHPS framework. The dataset contains WLAN, BLE and IMU sensor data in different orientations and multiple walking trajectories. Using this dataset, we showed how the process network can easily be adapted to optimise the behaviour of the positioning algorithms for symbolic space hit rate or the latency in a trajectory. In future work we plan to further increase the flexibility by supporting different types of users, ranging from developers to less-technical end users who want to use OpenHPS for asset tracking. While our symbolic spaces support multiple floors, we might further explore the challenges of performing floor detection in a building. Overall, OpenHPS has proven its flexibility in handling an indoor positioning use case with modular layers of abstraction such as symbolic spaces or high-level API endpoints.

## References


[1] K. A. Nguyen, Z. Luo, and C. Watkins, "Epidemic contact tracing with smartphone sensors," *Journal of Location Based Services*, vol. 14, no. 2, September 2020.

[2] M. Van de Wynckel and B. Signer, "OpenHPS: An open source hybrid positioning system," Vrije Universiteit Brussel, Tech. Rep. WISE-2020-01, December 2020.

[3] J. Hightower, B. Brumitt, and G. Borriello, "The location stack: A layered model for location in ubiquitous computing," in *Proceedings of WMCSA 2002*, Callicoon, USA, June 2002.

[4] A. Ranganathan, J. Al-Muhtadi, S. Chetan, R. Campbell, and M. D. Mickunas, "Middlewhere: A middleware for location awareness in ubiquitous computing applications," in *Proceedings of Middleware 2004*, Subotica, Serbia, October 2004.

[5] S. Pasricha, V. Ugave, C. W. Anderson, and Q. Han, "LearnLoc: A framework for smart indoor localization with embedded mobile devices," in *Proceedings of CODES 2015*, Amsterdam, Netherlands, October 2015.

[6] A. Bekkelien and M. Deriaz, "Hybrid positioning framework for mobile devices," in *Proceedings of UPINLBS 2012*, Helsinki, Finland, October 2012.

[7] M. Ficco and S. Russo, "A hybrid positioning system for technology-independent location-aware computing," *Software: Practice and Experience*, vol. 39, no. 13, September 2009.

[8] S. J. Barbeau, M. A. Labrador, P. L. Winters, R. Pérez, and N. L. Georggi, "Location API 2.0 for J2ME: A new standard in location for Java-enabled mobile phones," *Computer Communications*, vol. 31, no. 6, April 2008.

[9] R. Montoliu, E. Sansano-Sansano, J. Torres-Sospedra, and Ó. Belmonte-Fernández, "IndoorLoc platform: A web tool to support the comparison of indoor positioning systems," in *Geographical and Fingerprinting Data to Create Systems for Indoor Positioning and Indoor/Outdoor Navigation*. Academic Press, 2019.

[10] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: An open-source robot operating system," in *Proceedings of the International Workshop on Open Source Software*, Kobe, Japan, May 2009.

[11] T. Foote, "tf: The transform library," in *Proceedings of TePRA 2013*, Woburn, USA, April 2013.

[12] J. Hurtuk, J. Červeňák, M. Štancel, M. Hulič, and P. Fecilak, "Indoor navigation using indooratlas library," in *Proceedings of SISY 2019*, Toronto, Canada, September 2019.

[13] B. L. Decker, "World Geodetic System 1984," in *Proceedings of the 4th International Geodetic Symposium on Satellite Positioning*, Austin, USA, April 1986.

[14] W. Elmenreich, "An introduction to sensor fusion," Vienna University of Technology, Tech. Rep. 47/2001, November 2002.

[15] A. Popescu, "Geolocation API specification 2nd edition," November 2016.

[16] E. Zimányi, M. Sakr, A. Lesuisse, and M. Bakli, "MobilityDB: A mainstream moving object database system," in *Proceedings of SSTD 2019*, Vienna, Austria, August 2019.

[17] A. L. Davis, *Reactive Streams in Java*. Apress, 2019.

[18] M. Geilen and T. Basten, "Reactive process networks," in *Proceedings of EMSOFT 2004*, Pisa, Italy, September 2004.

[19] Y. Gu, A. Lo, and I. Niemegeers, "A survey of indoor positioning systems for wireless personal networks," *IEEE Communications Surveys & Tutorials*, vol. 11, no. 1, 2009.

[20] H. Liu, H. Darabi, P. Banerjee, and J. Liu, "Survey of wireless indoor positioning techniques and systems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 37, no. 6, 2007.

[21] H. Butler, M. Daly, A. Doyle, S. Gillies, T. Schaub, and T. Schaub, "The GeoJSON format," RFC 7946, August 2016.

[22] J. Hightower and G. Borriello, "Location systems for ubiquitous computing," *Computer*, vol. 34, no. 8, August 2001.

[23] M. Van de Wynckel and B. Signer, "OpenHPS: Single floor fingerprinting and trajectory dataset," May 2021. [Online]. Available: https://doi.org/10.5281/zenodo.4744379

[24] N. Fet, M. Handte, and P. J. Marrón, "A model for WLAN signal attenuation of the human body," in *Proceedings of UbiComp 2013*, Zurich, Switzerland, September 2013.

[25] A. Moreira, M. J. Nicolau, F. Meneses, and A. Costa, "Wi-Fi fingerprinting in the real world: RTLS@UM at the EvAAL competition," in *Proceedings of IPIN 2015*, Calgary, Canada, July 2015.