Indoor Positioning Using the OpenHPS Framework

Maxim Van de Wynckel, Beat Signer

Web & Information Systems Engineering Lab
Vrije Universiteit Brussel
What is OpenHPS?
An Open Source Hybrid Positioning System

Data Frame

Data frames are envelopes that are transmitted and processed through a positioning model. These frames are created by source nodes (e.g., sensors) and contain one or more data objects needed to process the frame.

A frame should contain a single reading of a sensor (such as an image of a video stream or current acceleration) and not permanent or calculated information.

Creating data frames

OpenHPS is a framework that processes sensor information to retrieve a position for one or more data objects. These objects are contained within an envelope called a data frame.

```python
import { DataObject, DataFrame } from '@openhps/core';

const myObject = new DataObject("Signature", "Great Singer");
const frame = new DataFrame();
frame.addObj1(myObject);

// method: DataFrame.addObject(object: DataObject): void
```

A basic data frame supports the addition of objects. Extended versions of this basic data frame also add additional sensor data.

Creating a custom data frame

Similar to data objects, decorators have to be used to indicate a serializable data frame.

```python
import { DataFrame, SerializableObject, SerializableReader } from '@openhps/core';

@SerializableObject()
export class MyDataFrame extends DataFrame {  
  public readonly test: any = undefined;
}
```
What is OpenHPS?

An Open Source Hybrid Positioning System

- Any technology
- Any algorithm
- Various use cases
- Flexible processing and output
  - Accuracy over battery consumption, reliability, ...
- Aimed towards
  - Developers
  - Researchers
Process Network Design

IMU Sensor
Process Network Design ...

IMU Sensor

... \rightarrow \text{IMU Source} \rightarrow \text{PDR Processing} \rightarrow \text{Merge Frames} \rightarrow \text{Display Position}

...
Process Network Design ...

IMU Source → PDR Processing → Merge Frames → Display Position

... → ...

Load Data Service

Save
Modularity

Abstractions (e.g. location-based services)

Communication (e.g. socket connection)

Positioning Techniques (e.g. fingerprinting)

Data Storage (e.g. MongoDB)

Core Component
Modularity ...

Communication
Socket, MQTT, REST API, ...

Data Storage
MongoDB, LocalStorage, RDF, ...

Positioning Algorithms
IMU, fingerprinting, OpenVSLAM, ...

Abstractions
Geospatial, location-based services, geojson, ...

Other
React-Native, NativeScript, Sphero, ...
Data Processing

Knowledge

Raw Data

Processed Data
DataObject
Absolute and Relative Positions

Absolute

► 2D, 3D, Geographical, ...

Relative

► Distance, angle, velocity, ...
► Relative to another object
DataFrame

VideoDataFrame

- **uid**: "camera"
- **position**:
  - \(x: 2, y: 5, z: 3\)
- **projection**: ...
- **width**: 1280
- **height**: 1024

DataObject

- Detected object

DataObject

- Detected object

DataObject

- Detected object

Image

Detected object

Detected object

Detected object
DataFrame ...

Pushing Data

SourceNode → ProcessingNode → ProcessingNode → SinkNode

- push(data)
- push(data')
- push(data'')
- resolve push
- resolve push
- resolve push
- completed data''
- completed data''
- completed data''

[Diagram showing data flow and processing nodes]
DataFrame ...

Pulling Data

SourceNode  ProcessingNode  ProcessingNode  SinkNode

pull()  pull()  pull()  
push(data)  push(data)  push(data)  
resolve pull  resolve pull  resolve pull  
resolve push  resolve push  resolve push  
completed data"  completed data"  completed data"  

DataFrame ...

Pushing Error

SourceNode → ProcessingNode → ProcessingNode → SinkNode

- push(data)
- resolve push
- push(data')
- resolve push
- push(data'')
- resolve push
- emit error
- emit error
- reject push
SymbolicSpace

An object that semantically defines a space

- Spatial hierarchy
- Graph connectivity with other spaces
- Geocoding
- GeoJSON compatibility
- Can be used as a location
- Can be extended ...
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),
  ]);
Location-based Service

currentPosition("me", ...)

Source -> Processing -> Fusion -> Sink

Storage

Location-based Service
Location-based Service ...

Source → Processing → Fusion → Sink → Location-based Service

setCurrentPosition("me", ...)

Source → Processing

Storage
Location-based Service ...

watchPosition("me", ...)

Source → Processing → Fusion → Sink

Source → Processing

Location-based Service

Storage
Demonstration

- Indoor positioning use case
- Use existing techniques
- Validation of flexibility and modularity
Positioning Model

- WiFi Source
- BLE Source
- User Input
- Merge Frames
- Socket Sink

Offline-stage App
Positioning Model ...

- Sink
- Fingerprint Storage
- Socket Source
- Socket Source
- WLAN Fingerprint Processing
- BLE Fingerprint Processing
- Fingerprint Service
- Beacon Data Service
- Server
Positioning Model ...

Socket Sink

WiFi Source

BLE Source

Merge Frames

IMU Source

PDR Processing

Position Fusion

Velocity Processing

Delay

Display Sink

Socket Source

Online-stage App

Velocity Processing
Positioning Model ...
Positioning Model ...

Online-stage App

1. WiFi Source
2. BLE Source
3. IMU Source
4. PDR Processing
5. Position Fusion
6. Velocity Processing
7. Display Sink
8. Delay
9. Velocity Processing
10. Socket Source
11. Merge Frames
12. Socket Sink
Positioning Model

Online App

PDR Processing

- SMA Filter: Simple moving average filter for acceleration
- Gravity Processing: Filter out gravity using orientation
- Step Detection: Adapt linear velocity based on detected step(s)
- Velocity Processing: Apply velocity to last known position
Positioning Model

Online App

```javascript
ModelBuilder.create()
  .addShape(GraphBuilder.create())
  .from(new IMUSourceNode({
    source: new DataObject(phoneUID),
    interval: 20,
    sensors: [
      SensorType.ACCELEROMETER,
      SensorType.ORIENTATION
    ]
  }))
  .via(new SMAFilterNode(
    frame => [frame, "acceleration"],
    { taps: 10 }
  ))
  .via(new GravityProcessingNode({
    method: GravityProcessingMethod.ABSOLUTE_ORIENTATION
  }))
```
Dataset

- Training data
- BLE Beacon
- Test data
Dataset ...

Total BLE Beacons: 11
Total detected WLAN access points: 220
Total stable WLAN access points: 199

<table>
<thead>
<tr>
<th></th>
<th>Training</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Datapoints</td>
<td>110</td>
<td>30</td>
</tr>
<tr>
<td>Total fingerprints</td>
<td>440</td>
<td>120</td>
</tr>
<tr>
<td>Duration (per orientation)</td>
<td>20s</td>
<td>20s</td>
</tr>
<tr>
<td>Avg. WLAN Scans (per fingerprint)</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Avg. BLE Advertisements (per fingerprint)</td>
<td>16</td>
<td>15</td>
</tr>
</tbody>
</table>
## Validation Results

### Static Positioning

<table>
<thead>
<tr>
<th></th>
<th>WLAN fingerprinting</th>
<th>BLE fingerprinting</th>
<th>BLE multilateration</th>
<th>Fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>failed points</strong></td>
<td>0</td>
<td>6</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td><strong>average error</strong></td>
<td>1.23 m</td>
<td>3.23 m</td>
<td>4.92 m</td>
<td>1.37 m</td>
</tr>
<tr>
<td><strong>minimum error</strong></td>
<td>0.01 m</td>
<td>0.17 m</td>
<td>0.74 m</td>
<td>0.01 m</td>
</tr>
<tr>
<td><strong>maximum error</strong></td>
<td>4.77 m</td>
<td>15.39 m</td>
<td>19.26 m</td>
<td>9.75 m</td>
</tr>
<tr>
<td><strong>hit rate</strong></td>
<td>95.82 %</td>
<td>80.83 %</td>
<td>52.50 %</td>
<td>96.67 %</td>
</tr>
</tbody>
</table>
Validation Results ...

Trajectories

- Sensor fusion
- WLAN & BLE Cell-ID
- Expected trajectory
- Trajectory start

OpenHPS
### Validation Results...

#### Trajectories

<table>
<thead>
<tr>
<th></th>
<th>WLAN + BLE</th>
<th>WLAN + BLE + IMU</th>
</tr>
</thead>
<tbody>
<tr>
<td>average error</td>
<td>3.28 m</td>
<td>1.26 m</td>
</tr>
<tr>
<td>maximum error</td>
<td>9.60 m</td>
<td>3.10 m</td>
</tr>
<tr>
<td>average update frequency</td>
<td>3.04 s</td>
<td>0.52 s</td>
</tr>
</tbody>
</table>

- **Sensor fusion**
- **WLAN & BLE Cell-ID**
- **Expected trajectory**
- **Trajectory start**
Contributions and Conclusions

- OpenHPS: **open source** framework for hybrid positioning
  - Aimed towards **developers** and **researchers**
- **Abstractions** such as location-based services and spaces
- Validation of an indoor positioning use case
- Configurable and interchangeable **nodes** and **services**
- **Public dataset** with multiple orientations

Visit [https://openhps.org](https://openhps.org) for additional resources, documentation, source code and more!