A Hybrid Localization Framework Supporting Multiple Standards and Manifold Post-Processing

Marco Gunia*, Bo Zhang*, Niko Joram* and Frank Ellinger*
*Chair of Circuit Design and Network Theory (CCN)
Technische Universität Dresden, 01062 Dresden
Email: marco.gunia@tu-dresden.de

Abstract—Nowadays much research activities are devoted to provide hybrid localization. Software written for systems based on a single standard is usually highly customized due to the differences of the underlying technologies. The reasons can already be seen by comparing the input and output parameters. However, the limitation to only a few underlying standards leads also to a high degree of specialization for hybrid systems. Another problem is time-to-market, as a result of which software is developed rather quickly. Both results in software, which is neither variable nor extensible. To take into account these problems, this paper proposes a framework for hybrid localization. After identifying drawbacks of related approaches, requirements are compiled. In contrast to most publications, this paper puts special emphasis on implementation details. As a novelty, our framework is based on operators, which enables to treat generation and processing of data in an equal manner. Hereby, many real world problems, like coordinate transformations, can be solved naturally. Moreover, our framework is capable of dealing with arbitrary input and output parameters and it supports push and pull behaviour. We test our solution by applying diverse received signal strength-based algorithms. Experiments and simulations are performed to show the potential of the framework and its broad application.

I. INTRODUCTION

Localization is becoming one of the key criteria for selecting modern embedded systems. On the one hand, this is visible due to the integration of positioning technology in today’s systems, e.g. the inclusion of Global Positioning System (GPS) hardware in present-day mobile phones. Moreover, communication signals can be applied for positioning, e.g. Bluetooth or Wireless Local Area Network (WLAN). On the other hand, new applications have been created, which more strongly rely on the current position, for instance for guidance and gaming.

To build up a fully working system, various challenges have to be tackled. Much effort is devoted to hardware aspects, e.g. integration issues like miniaturization or battery consumption. Due to the pressure of time-to-market, software for embedded systems is often produced rather quickly. This results in libraries, which are neither variable nor extensible. The term variability is used to denote the level of support to varying behaviour, e.g. the transfer of the software to a new platform. In contrast, extensibility indicates the level of support to integrate additional features, e.g. new hardware entities.

Embedded systems offering support for localization are usually targeted to one or a few underlying signals. Many single-signal systems exist, each exhibiting different pros and cons. In contrast, some systems deal with the combination of multiple localization information to generate superior estimates. Seamless transition between signals is a related field, taking into consideration their availability and quality.

Software written for these systems is highly customized, due to the individuality of the localization concepts [1]. This becomes evident from the fact that each system depends on various different parameters. For instance, Inertial Navigation Systems (INS) require only initial conditions, i.e. starting point, initial velocity and orientation, whereas algorithms for secondary radar have to be provided with the positions of the base stations. Depending on the underlying channel model, additional information is specified, e.g. parameters for the log-normal channel model for Received Signal Strength- (RSS) based systems. Consequently, source code can hardly be transferred. Similarly, sensor fusion is highly tailored to the underlying signals, making it difficult to support alternatives.

In this paper, a framework kernel is introduced for variably incorporating multiple localization sources, which are possibly based on different input parameters and preconditions. Operators represent the core concept, whose introduction enables to treat data provision, data processing and data filtering in a unique manner. In doing so, results from various sources, algorithms or filters can be flexible combined to generate improved localization estimates. Moreover, this enables to naturally solve many real-world issues, e.g. coordinate system transformation, which were pointed out as a problem in [2]. Hereby, data processing techniques based on the measurement data for the current time are denoted as algorithms. In contrast, filters determine an estimate by additionally using prior data. An example for the latter would be a Kalman-filter.

The rest of this paper is organized as follows. Section II presents related work. Requirements for the framework are compiled in section III. This is followed in section IV by an introduction into the design patterns, which form the foundation of the framework. The framework itself is developed in section VI. However, firstly there will be a presentation of different RSS positioning algorithms for WLAN in section V. As a means to demonstrate the concepts, these are freely combined in section VII to show the potential of the framework. The last section VIII concludes the paper.

II. RELATED LITERATURE

A comparative study of multiple localization systems is given in [3]. The authors underline that many works are
focussed on parameters like accuracy, power, precision, infrastructure, performance and costs. In addition, a hybrid framework to optimize positioning is presented in this reference. However, the system is very restricted, since it employs WLAN for coarse localization and RFID tags to refine the estimate. Attempts to support additional signals or filters are not made. Generally speaking, software implementation aspects are usually only mentioned briefly in the literature. Even papers especially concentrating on implementation issues, e.g. [4] or [5], are mostly dealing with implementation aspects towards increasing accuracy or reducing efforts.

A survey of sensor fusion architectures is presented in [6], ranging from abstract to rigid architectures. Many approaches are based on layers with fixed interfaces, e.g. the abstract fusion model originating from the US Joint Directors of Laboratories (JDL) [7]. The architecture of JDL consists of five levels, build on one another. Incoming sensor data is first pre-processed (level 0), coordinate system transformed (level 1) and used for contextual description of the scenario (level 2). With the help of these results, inferences can be drawn (level 3). Besides, system performance is monitored (level 4). As highlighted in [6], variability and extensibility is limited. An example for a rigid layered approach with well-defined interfaces for hybrid indoor geolocation is found in [8]. Compared to our framework, it is restricted to indoor localization and does not support nesting behaviour. Moreover, it only considers algorithms, whereas our approach supports general operators, which enables to utilize data processing entities like filters in an equal manner as algorithms. In summary, all layered-based concepts are somehow restricted, since they do not support arbitrary nesting and need to define new interfaces for each new data source. Cycle-based approaches, introduced in [6], are alternatives. For instance, the Boyd cycle consists of four repeating steps: Observation, orientation, decision and action. Compared to the layered concepts, the model provides more degrees of freedom. However, it is an abstract concept, which is not specifically tailored to localization.

There are many other localization frameworks, e.g. [1], [2], [3]. One worth mentioning is the Global Positioning Module (GPM) [9]. This framework seamlessly combines various concepts to support indoor and outdoor positioning. Position providers are classified by their performance properties, which enables the selection based on their suitability (i.e. accuracy, costs or battery consumption). Except pre-processing, it does neither support fusion of different inputs nor nesting behaviour. As an example, seamless transfer is implemented by means of the simplest approach, i.e. switching the source.

In contrast to the presented techniques, our approach is based on operators. Complex operators can be composed of simple ones. Localization algorithms, filters or other data processing entities can be considered as basic elements. By doing so, cumbersome real-world issues like coordinate system transformation or data sampling can be incorporated naturally. Moreover, our framework permits to specify arbitrary localization parameters, independent of the underlying operator. The latter is a novelty, introduced in section VI.

III. METHODOLOGY

Our approach is intended to be an universal framework, which could be easily applied to a multitude of localization problems by performing only minor adaptations. In addition, it should support arbitrary localization signals, as well as fusion and filter techniques. General demands for localization frameworks are introduced in [2] and [9]. With the help of these preliminary works, the following essential requirements have been identified:

- **Variability**: Localization algorithms for different signals are very diverse, which is visible by various input parameter sets, localization results or underlying coordinate systems. For instance, no input parameters have to be passed to GPS mobile devices by the user and the result is returned in global coordinates as longitude and latitude. In contrast, INS require the specification of the initial position/orientation and might return the new position/orientation in Cartesian or global coordinates. Hence, the framework needs to cope with varying behaviour.

- **Extensibility**: New localization signals should be included without any code modifications to existing application programs. Moreover, integration into the framework has to be achieved with little expense. Likewise, simple adding of data processing elements is preferable.

- **Uniform interface**: Framework elements, e.g. algorithms, fusion entities or noise filters, are to be called in a uniform way and have to return the result in a standardized format.

- **Nesting behaviour**: Arbitrary nesting of basic framework elements should be supported to include as many diverse localization approaches as possible. An example is INS, where accelerometer data from one sensor could internally already be noise-filtered, whereas a system from a different manufacturer only provides raw-data. To uniformly support both devices, a Kalman-filter might be included on the very low level for the second device, even though another Kalman-filter is used for sensor fusion between accelerometer and gyroscope at a higher level.

- **Push / Pull behaviour**: Push behaviour specifies the explicit invocation of a calculation by the user, whereas the re-triggering of the calculation automatically by the system due to an event is denoted as pull behaviour. Both alternatives are to be supported.

- **Platform independence**: The framework should be easily transferred to a new platform. In particular, this is important due to the availability of the three main platforms, i.e. Googles Android®, Apples iOS® and Microsofts own Windows Phone®.

IV. DESIGN PATTERNS

Design patterns represent descriptions of standard solutions for frequent occurring design problems. These are easy and smart approaches, which are usually not found from the beginning, but which are characterized by providing additional flexibility [10]. Many of the patterns treat variability and extensibility, hence they are perfectly suited to be a key step regarding meeting the requirements from section III. In the
following, four patterns for object oriented programming are introduced, which are the foundation of the framework.

**Strategy** (Fig. 1) denotes a variability pattern to individually select an algorithm from a family of related algorithms [10]. The Concrete Strategy classes implement these individual algorithms, whereby their interface is defined by the abstract class Strategy. Context possesses a reference to an object of this class, whereas the concrete object referred could either be of the Concrete Strategy. Thus, the algorithm invoked by Context could be changed at run-time.

**Composite** (Fig. 2) is an extensibility pattern which allows creating hierarchies for complex objects based on single elements. Clients are enabled to deal with single and complex objects equally, disregarding their differences. Leafs represent the single elements, which inherit their interface from Component. Equally, a Composite exhibits the same interface, but additionally has a link to another Component, which could be either a Leaf or a Composite. Hence, whole hierarchies are created [10].

**Extension Object** (Fig. 3) is a complex extensibility pattern, dealing with Extensions as optional parts. Extensions can be added, utilized or removed by the clients. Therefore, Subject offers an interface to handle extensions, which is inherited by any Concrete Subject. This interface enables to treat all extensions equally, even though they might completely differ. The client or the Concrete Subject can then perform actions based on the availability or attributes of Concrete Extensions [11].

**Observer** (Fig. 4) is a behaviour pattern to inform objects, which depend on the state of other objects, about changes. The Concrete Observer represents the dependent object, which is informed by the Concrete Subject about changes. Initially, observers have to register at the subjects by calling register(). Afterwards, state changes are published by the subjects calling refresh() on the observer.

<table>
<thead>
<tr>
<th>Client</th>
<th>Operation()</th>
<th>AddComponent()</th>
<th>RemoveComponent()</th>
<th>ReturnChildObject()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concrete Component</td>
<td>Operation()</td>
<td>AddComponent()</td>
<td>RemoveComponent()</td>
<td>ReturnChildObject()</td>
</tr>
<tr>
<td>Childreen</td>
<td>Concrete Component</td>
<td>Operation()</td>
<td>AddComponent()</td>
<td>RemoveComponent()</td>
</tr>
</tbody>
</table>

**V. RSS ALGORITHMS**

To highlight the capabilities of our framework, developed in section VI, multiple diverse localization algorithms are introduced in the following. We limit our considerations to techniques based on the RSS, since these concepts already reveal a wide variety, which is sufficient to demonstrate the characteristics of our framework.

In the remainder of this section common algorithms are introduced. A subset of these is based on assumptions regarding the underlying channel model. Although, multiple models have been described, due to its simplicity, the log-normal channel model is the most widely used. It is described next and it is the foundation for some of subsequent algorithms.

The log-normal model relates the distance between receiver and transmitter $d$ to the received signal power $P_{RX}$ [12],

$$P_{RX}/dBm = A - 10 \cdot \eta \cdot \log_{10} \frac{d}{d_0} + \mathcal{N}$$  \hspace{1cm} (1)

$A$ is a constant, which depends on the transmitting power, the gains of the sender and receiver antennas, and the received power at distance $d_0$. By means of the path loss coefficient $\eta$, environmental conditions can be taken into account. Moreover, random effects are included by $\mathcal{N} \sim \mathcal{N}(0, \sigma^2)$, which is a Gaussian random variable with zero mean and variance $\sigma^2$.  
A. Non-calibrated algorithms

Non-calibrated variants do not make any assumptions about the underlying channel model. Thus, they are simple. However, a well-known channel model contains additional information regarding the environment, which could be used to improve accuracy. However, the channel model needs to properly reflect the real situation. Otherwise, accuracy could be degraded heavily, e.g. due to fading effects. Two simple approaches not making use of the concrete channel model are Proximity and Centroid. In the former, the position of the sender with the highest received signal power is used as an estimate for the position. In contrast, all base stations, which are above a certain threshold are involved in the latter approach by calculating their center. As an improvement, n-Centroid only considers the positions for the n transmitters providing the signals at the receiver with the highest power. A sophisticated approach, called Ecolocation [13], is based on the subdivision of the scenario in disjoint sectors. Each possible measurement RSS vector (in the vector each element represents the RSS from one base station) could then be distinctly assigned to one of these areas. The final estimate is the center of gravity of the selected sector. Another technique is Fingerprinting, where the RSS values in different spots in the environment are scanned offline and stored in a database. In the online phase, the measurement is then compared to the entries in a database and the best match is selected as an estimate.

B. Calibrated algorithms

Calibrated algorithms are based on the precise knowledge of the parameters A and η from equation (1), by prior calibration. The Circular algorithm focusses on minimizing the sum of least squares by using a gradient approach to seek for the optimum. In contrast, for the Hyperbolic technique the solution is reduced to a linear equation system, which is then solved by the least squares method. Both algorithms can be enhanced by weights. These Weighted Circular and Weighted Hyperbolic generally perform better as their counterparts, since the values of RSS measurements, which are far away, are taken into account less [12]. An alternative to performing a gradient technique for Circular and Weighted Circular, which could not converge, is discretization of the underlying scenario. A subsequent grid search then reveals the local minimum. We call this Simple (Weighted) Circular.

In summary, Proximity and Centroid only depend on the locations of the base stations and the RSS measurement vector to perform an estimation. For Ecolocation the boundary of the scenario has to be specified additionally. All calibrated variants require the definition of the parameters for the log-normal model. Moreover, Circular and Weighted Circular demand the definition of the break condition, i.e. the maximum error or the maximum iteration steps. Furthermore, a discretization distance is to be specified for the Simple (Weighted) Circular. For Fingerprinting, a database of RSS vectors must be available for calculating the estimate. Thus, all algorithms request for various diverse input parameters.

VI. Framework

Fig. 5 illustrates the architecture of the framework kernel. To highlight the individual influences from the design patterns, the colours of the classes involved have been selected in accordance to Fig. 1, 2, 3 and 4. The class Operator plays a key role, since it represents an arbitrary data processing entity, e.g. a localization Algorithm or a Filter. The latter may be a component for sensor fusion, noise reduction or coordinate transformation. The execution is invoked by calling doCalculation() on the Operator. Each Operator provides an interface to receive Extensions, which contain ancillary data to perform the operation. Hence, these are supposed to be linked before starting the execution. As an example, extensions may be channel model parameters, descriptions of the underlying scenario, a database for storing fingerprints, etc. These data must be prepared such that it is a subclass of the abstract class Extension, respectively. Each concrete operator individually takes care about the extensions. This means that each operator should be implemented in such a way, that the availability of the appropriate extensions is checked before performing any calculations. Consequently, our framework throws an exception, if an extension is missing. This exception must then be handled by the caller, possibly by adding the missing data and restarting the operator. Moreover, arbitrary nesting of operators is enabled by providing a link back to the Abstract Operator. Hence, measurement data can either represent a basic operator, nested within the operator, or it could be linked as extension, e.g. as extension to a localization algorithm.

Fig. 6 shows an example for applying the framework. There are three algorithms A1, A2 and A3, which have links to some data D1, D2 or D3, respectively. These data is provided by internal operators. The results of A1 and A2 are applied to two filters F1 and F2, e.g. for noise reduction, coordinate transformation or tracking. These outputs together with the result of A3 are then forwarded to F3, e.g. a Kalman-filter responsible for fusioning the results. Each of the filters may
have connected extensions, e.g. the covariance matrices $E_5$ and $E_6$ for the Kalman-filter. $F_1$, $F_2$ and $A_3$ are linked as extensions to $F_3$. Push behaviour can simply be realized by calling the $doCalculation()$ method on the outer operator $F_3$. Within this method further $doCalculation()$ methods of dependent operators are invoked subsequently. In this example, one drawback of the framework is visible: Only one operator can be nested each time. Hence, the results of $F_1$, $F_2$ and $A_3$ must be made available to $F_3$ using extensions.

![Fig. 6: Framework Example](image)

To support pull behaviour, operators implement both the methods from Subject and from Observer from Fig. 4. In doing so, operators can trigger dependent operators, in the case of data becoming obsolete. As an example, imagine algorithms $A_1$, $A_2$ and $A_3$ from Fig. 6 to be dependent on an operator representing RSS measurements. Once new measurements are available, the RSS operator can inform the algorithms, which redo their calculations. Equally, the filters are informed by the algorithms, and so forth. Since $F_3$ links its dependent operators via extensions, it does not redo its calculation using the method described. However, full pull behaviour is supported, if class Extension from Fig. 5 is inherited from the class Operator. Note, that for pull behaviour the trigger for re-calculation is an innermost operator, whereas it is the outermost operator for push behaviour.

Referring to the requirements in section III, the framework offers a high degree of variability. On the one hand, this can be traced back to the extensions, which enable supporting diverse input parameters. Besides measurement data, this could be additional information like simple constants or complex databases. On the other hand, the return value of the method $doCalculation()$ of each operator is again an object of the class Operator. As a standard interface, it allows to nest operators. Hence, the uniformity and nesting requirements from section III are fulfilled. A high degree of extensibility is achieved by keeping the system structure universal, facilitating coupling of alternative algorithms. This paper focusses mainly on RSS-based concepts, but the framework can easily be extended to include time of arrival-, time difference of arrival- or angle of arrival-based systems. That is due to the application of the Strategy and Extension Object pattern. Explicit triggering of a calculation is equally possible, as well as event driven re-calculation, thus providing both push and pull behaviour.

Since Google’s Android®, Apples iOS® and Microsofts own Windows Phone® all support Java, the framework can be transferred to each of the three today’s main platforms. It represents an independent and stand-alone framework, which runs on a single device. It does not depend on external hardware, although external devices can be connected. In this way, platform independence is guaranteed. Moreover, activation via Python scripts is possible, which enables to use the same code for simulations as is used on a real device. Simulations are performed using this interface in the next section.

**VII. Experiments**

This section focusses on demonstrating the benefits of the framework using simulations and experiments. Here, we pay less attention on the actual results, but rather on showing the capabilities. In a first step, it is invoked by a Python script to make conclusions about the expected error characteristics of the algorithms described in section V. Our scenario is a room of size $10 \times 10$ m, with the base stations situated in the corners. We have discretized the area, calculated the ideal RSS values using equation (1), added noise, and used these values as inputs for all the algorithms. The noise is Gaussian $\mathcal{N} \sim \mathcal{N}(0, 1)$ with zero mean and variance 1. Hence, we suspect $3\sigma$, i.e. 95% of the RSS measurements, to be within $\pm 3$, a realistic assumption which is supported by our measurements. All steps have been implemented on the basis of the framework, using operators. We define the error $\varepsilon$ as the difference of actual position $[x_a, y_a]^T$ and estimated position $[x_e, y_e]^T$, i.e.

$$\varepsilon = \sqrt{(x_a - x_e)^2 + (y_a - y_e)^2} \quad (2)$$

Fig. 7 presents the results. In a subsequent paper, where we will develop metrics to estimate the localization quality, these outcomes will be discussed in detail. Here, it is sufficient to note, that the results show mainly the expected behaviour. For the non-calibrated variants the dependence of the error on the actual position is obvious. Proximity offers the smallest error in the corners near the base stations, whereas the error is minimal in the middle of the room for Centroid. The subdivision of the underlying area by disjoint polygons as described in [13] for Ecolocation leads to a more uniformly distributed error. All calibrated variants show independent error characteristics, expect for Weighted Circular. Even though this algorithm was pointed out to be superior in [12], the figure demonstrates that this statement highly depends on the position.

To illustrate the application of the framework, we have implemented a program for tracking on an Android platform, which is shown in Fig. 8. A part of it is an intuitive drawing algorithm to quickly define the scenario. Afterwards, the framework is invoked by either push or pull behaviour to track the position. Both variants have been implemented and are working. As an example, for pull behaviour the RSS measurements are triggered automatically by the Android platform. This represents an innermost operator returning the distance by using equation (1). The calibration constants have been determined in a separate experiment beforehand. This operator is nested within each of the RSS algorithms, presented in...
is an operator, having the following characteristics: Arbitrary underlying signals and data processing entities like filters are supported, where both elements are treated uniformly. Amongst others, this enables to create complex operators out of simple ones. With the help of extensions, the interface is held variable to support various types of operations. Thus, real world problems like coordinate transformations or time synchronization between different sources can be included naturally. One drawback of our framework is the limitation to only nest one operator at a time. However, this can be alleviated by the inclusion of further operators as extensions. The conducted experiments show the potential of the framework with the example of RSS-based algorithms. The ideas remain true for other approaches, e.g. time-difference-of-arrival-based radar, and the transfer is straightforward.

Future work will concentrate on defining characteristics of our operators to create an evolutionary framework, where the structure of operators is automatically reordered to react to modified conditions. This could be changing power consumption requirements, e.g. a device switching to battery supply, or outdoor / indoor scenarios.

ACKNOWLEDGEMENT

The research leading to these results has received funding from the European Communitys Seventh Framework Programme (FP7/2007-2013) under grant agreement ICT-FP7-611526 (MAGELLAN).

REFERENCES