IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON DISTRIBUTED COMPUTING INFRASTRUCTURE FOR CYBER-PHYSICAL SYSTEMS

Received August 16, 2019, accepted September 11, 2019, date of publication October 15, 2019, date of current version November 5, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2947581

Online Offline Learning for Sound-Based Indoor Localization Using Low-Cost Hardware

RÜDIGER MACHHAMER[1], MATTHIAS DZIUBANY[1], LEVIN CZENKUSCH[1], HENDRIK LAUX[2], ANKE SCHMEINK[2], (Senior Member, IEEE), KLAUS-UWE GOLLMER[1], STEFAN NAUMANN[1], AND GUIDO DARTMANN[1], (Senior Member, IEEE)

[1]Institute for Software Systems (ISS), Trier University of Applied Sciences-Environmental Campus Birkenfeld, 55761 Birkenfeld, Germany
[2]ISEK Research and Teaching Area, RWTH Aachen University, 52062 Aachen, Germany

Corresponding author: Rüdiger Machhamer (r.machhamer@umwelt-campus.de)

This work was supported in part by Federal Ministry of Education and Research under Grant 01IS17073. Sourcecode and data is available at https://cosy.umwelt-campus.de/software. Parts of this work are based on the master thesis of the first author. Special thanks to Diana Machhamer for the graphical abstract and the visualisations.

ABSTRACT Online Learning algorithms and Indoor Positioning Systems are complex applications in the environment of cyber-physical systems. These distributed systems are created by networking intelligent machines and autonomous robots on the Internet of Things using embedded systems that enable the exchange of information at any time. This information is processed by Machine Learning algorithms to make decisions about current developments in production or to influence logistics processes for optimization purposes. In this article, we present and categorize the further development of the prototype of a novel Indoor Positioning System, which constantly adapts its knowledge to the conditions of its environment with the help of Online Learning. Here, we apply Online Learning algorithms in the field of sound-based indoor localization with low-cost hardware and demonstrate the improvement of the system over its predecessor and its adaptability for different applications in an experimental case study.

INDEX TERMS Fingerprint recognition, incremental learning, indoor localization, internet of things, learning vector quantization, machine learning, online learning, signal processing.

I. INTRODUCTION

This paper presents the further development of an innovative Indoor Positioning System (IPS). IPS gain more and more importance with the increasing development of the Internet of Things (IoT). Industrial applications such as

- autonomous robots in smart factory environments,
- warehouse management applications for large devices or
- automated repacking stations

require information about positions of participating components in a life-long learning [1], [2] application to enable AI resource-conserving optimization. The quality of the position information such as accuracy, real-time capability and efficiency is directly related to the possibility of saving resources. The necessary costs for purchasing and operating the required devices influence the cost-effectiveness of the applications and thus their acceptance by industrial companies. In [3], accuracy and costs of the most popular IPS are compared. The paper concludes that there is still no satisfactory overall solution because accurate solutions are expensive or cannot localize in real time while cheap IPS are inaccurate. Furthermore, most recent technologies are based on triangulation and therefore require multiple, often expensive, devices. In addition, centimeter-scale technologies are either expensive or prone to failure. In this paper, we present an alternative technology that promises to yield outstanding results with cost-effective hardware aiming at replacing existing technologies in the long run, which often require high maintenance and acquisition costs while at the same time offering medium quality services. Regarding the required hardware, we only use a single $2 electronic microphone, a single ordinary loudspeaker and the WiFi-capable *ESP8266* based IoT-Kit *Octopus* [4] and improve localization performance by combining efficient Machine Learning (ML) algorithms like *K*-Nearest-Neighbor (*K*NN), *K*-Means and Learning Vector Quantization (LVQ) in a distributed cloud application. We apply these algorithms to room impulse response (RIR) records, which depend on the position of the microphone, the room geometry and the setup of the room. If the environmental conditions remain constant, the suitability of the RIR for localization becomes apparent.

Figure 1 schematically shows the relationship between the RIR and the position of the recording device. As the microphone moves away from the loudspeaker, the time required

The associate editor coordinating the review of this manuscript and approving it for publication was Zhen Ling.

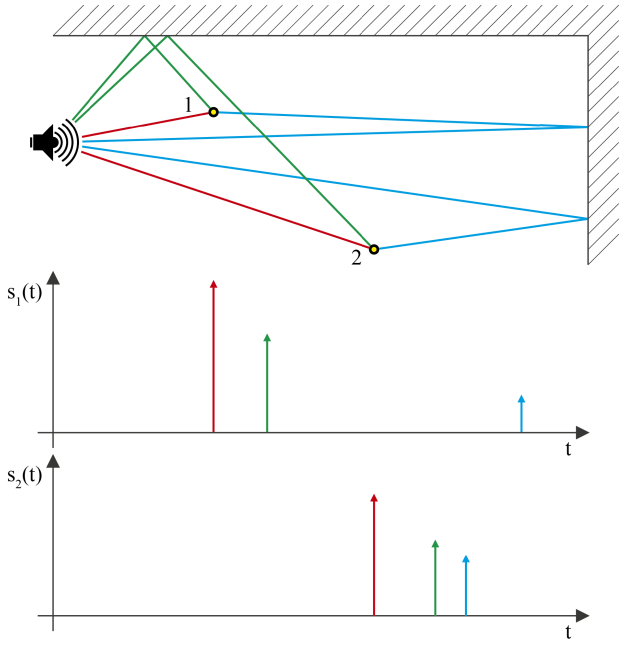

FIGURE 1. Schematic illustration of the room impulse response, the direct sound (red) reaches the microphones earlier with a higher amplitude than the first reflections (green and blue) due to signal propagation theory [4].

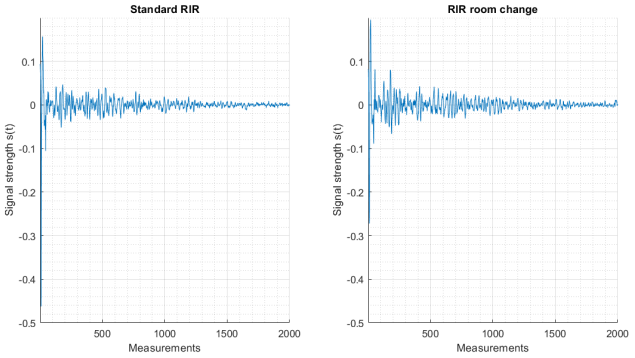

FIGURE 2. Comparison between standard RIR and RIR after room change.

for the signal to reach the microphone increases and the measured intensity decreases. The *Direct Sound* (red impulse) reaches the microphone first, the *First Reflections* (green and blue impulse) are received later and weaker due to the larger distance and the energy loss due to acoustic reflection. The locations of microphones 1 and 2 can be distinguished based on the different RIR $s_1(t)$ and $s_2(t)$.

Figure 2 shows two RIRs for the same position, while the right RIR is affected by a major room change. The plots show 2000 measured values recorded over a period of 0.2 seconds at a sampling rate of 10 kHz. Based on this information, a rule-based algorithm could be used to locate the microphone, but it would be highly susceptible to interference if changes were made in the room. Therefore, we investigated the use of algorithms of Unsupervised Learning (UL) and Supervised Learning (SL) that use Online Learning (OL) at runtime to adapt to changes in the localization environment.

A. RELATED WORK

We already demonstrated the applicability for static room setups [4] in 2018, where we extend a nature-inspired method [5] without need for high spatial accuracy or big microphone arrays which is inspired by [6] and [7] and uses *K*-Means generated representative prototypes in a *K*NN model. With this model, an 88 % success rate can be achieved in distinguishing 16 squares with a side length of 15 cm on a 60 cm × 60 cm table top. Since the classification relies on the RIR, which depends not only on the position of the microphone but also on the room geometry and the objects in the room, this method only works in very similar room configurations thus only being conditionally suited for real applications.

In order to counter these problems, a learning algorithm is used to adapt to room changes at runtime. Therefore, a combination of batch learning and OL [8] based on LVQ is used to address the problem of the stability plasticity dilemma [9], [10], which describes the problem of a learning system to preserve acquired knowledge while at the same time new knowledge is being built up. In this *offline online learning architecture* (OOLA), basic knowledge is acquired to achieve stability in a batch learning process before runtime (offline). In the case of sound-based indoor localization (SBI), this is achieved by initially learning the RIR from different positions in space using fingerprinting procedures. In order to realize the plasticity at runtime, the algorithm learns step by step, which is realized by the use of LVQ (online). The combination of the procedure is performed by a dynamic selection strategy, which uses information from both knowledge bases for a decision.

B. CONTRIBUTION

We extend the functionality of our IPS with low-cost hardware for more complex environments and improve the quality of positioning in static environments. Therefore, we adapt the described OOLA to the problem of our RIR based IPS and use LVQ to improve our initial prototypes depicting the classification model for static room setups. Further, we classify our system among other IPS.

C. STRUCTURE

Section II gives an overview of existing IPS and classifies ours regarding the given criteria. It gives a short overview of the applied algorithms and a simulation of their behaviour. Section III describes the components of the learning architecture and reviews the functionality by a case study of our IPS. Section IV summarizes the results and gives an outlook on further developments.

D. NOTATION

Vectors are indicated by bold letters $\mathbf{x}$, matrices by bold capital letters $\mathbf{M}$, transpositions are indicated by $\mathbf{x}^T$ or $\mathbf{M}^T$. Sets are marked with calligraphic font, e.g. $\mathcal{S}$, their cardinality is expressed by $|\mathcal{S}|$.

II. IPS AND SIMULATION

In this section, we give a short overview about existing IPS and present our method, further we describe simulations of the used algorithms.

A. IPS

The survey [3] compares existing IPS in 2017. They evaluate current approaches based on audible sound with an accuracy in the range of meters as inaccurate. First positioning experiments by evaluation of the RIR provide promising results in the decimeter range [4]. Other presented methods allow more precise positioning, but require a high amount of resources in the form of high-quality hardware or complex algorithms. The required routers for WiFi-based methods are usually available anyway, but must still be included in the resource calculation. Our approach only requires a few inexpensive devices and computers of average performance:

- a MAX4466 microphone to be located,
- an ordinary speaker giving the signal,
- a usual ESP8266 based controller board,
- and a common i5-laptop.

The survey [3] proposes a classification of the IPS according to the following three main criteria:

1) *signal carrier* - distinction between radio, light, sound or magnetic waves
2) *signal processing* - distinction between active and passive systems
3) *signal structure* - distinction between signals with and without embedded information.

If the mobile device receives a signal and the signal is processed by peripheral devices, this is referred to as active signal processing. Passive systems are characterized by mobile devices that process a received signal to determine their own position. Since the SBI uses the RIR, the mobile device receives a signal sent by the infrastructure, which is a criteria for passive signal processing. However, since this signal is evaluated after a transmission in the periphery, the procedure is classified as active signal processing. Signals with embedded information contain coded information, which can be used to estimate the location, e.g., device codes in light bulbs or time stamps in GPS [3]. Accordingly, the present methodology can be categorized as

1) a sound based locating method with
2) active signal processing
3) without embedded information.

The technique used in this work can be described as a combination of *fingerprinting* and evaluation of the signal propagation behaviour. Fingerprinting is already used in applications with WiFi, Bluetooth or magnetic waves. At first, a map of signal recordings (fingerprints) is created. This map allows conclusions to be drawn about the position by comparing stored data with the data to be localized [11]. Frequently Kalman or particle filters are used to filter out unwanted interference through the suppression of superimposition and noise [12]. Signal propagation deals with the behaviour of signals during their propagation in space. The strength of the signal decreases with distance, walls and obstacles lead to reflections and interference. The IPS discussed in this article uses this signal behaviour to distinguish different positions by means of automatic learning procedures. To the best of our knowledge, there is no similar methodology for indoor positioning. The listed sound-based methods mainly work via direct evaluation of signal propagation, mostly using multilateral methods that rely on signal propagation durations (e.g., *time of flight*) for position determination. A passive method [13] based on fingerprinting of audio recordings compares background noise from rooms and is not comparable in its accuracy. It is stated in [3] that no satisfactory solution has yet been found in the area of interior location. Precise technologies are too expensive or do not provide results in real time.

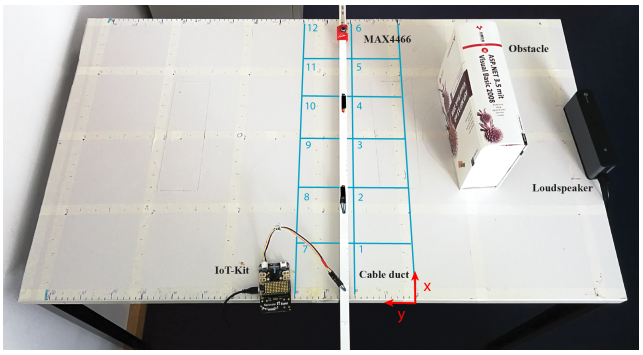

FIGURE 3. Experimental setup, the twelve blue squares depict the locating range, the book represents the obstacle which changes the RIR of the room. The measurements are carried out along the cable duct (x-direction) centimetre by centimetre row by row (y-direction).

B. LOCALIZATION SCENARIO

The core of the experiment is the localization of a microphone on a table using ML methods. The data required for this is generated by a loudspeaker repeatedly transmitting the same signal, which is recorded by a microphone at different positions on the test table. Figure 3 shows the table used with a 90 cm × 60 cm table top. The loudspeaker is fixed on the right side of the table and transmits its signal over the locating range in the direction of a reflecting wall to the left. To avoid possible ambiguity due to room symmetry, the loudspeaker is positioned at an angle of approximately 10° to the reflecting wall. The locating range corresponds to a 20 cm × 60 cm area 30 cm in front of the loudspeaker, which is divided into twelve squares with an edge length of 10 cm. The experiment is carried out in an ordinary office of 6 m × 3 m × 2.5 m size, in which furnishings for two persons are placed. The aim of the SL algorithm is to assign the distinctible recordings of the microphone to one of the twelve areas in order to estimate the position of the microphone. To do this, the MAX4466 microphone is placed centimetre by centimetre in each of these twelve fields to produce a labeled comparison recording (*fingerprinting*). To ensure uniform alignment across all recordings and to improve positioning

TABLE 1. Order of recordings related to position and label of area.

Number	Length x [cm]	Width y [cm]	Label
$1, 2, 3$	1	1	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$28, 29, 30$	10	1	1
$31, \ldots, 60$	$11, \ldots, 20$	1	2
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$151, \ldots, 180$	$51, \ldots, 60$	1	6
$181, \ldots, 360$	$1, \ldots, 60$	2	$1, \ldots, 6$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$1621, \ldots, 1800$	$1, \ldots, 60$	10	$1, \ldots, 6$
$1801, \ldots, 3600$	$1, \ldots, 60$	$11, \ldots, 20$	$7, \ldots, 12$

accuracy, the microphone is mounted on the sliding cover of a cable duct. This cable duct can be moved parallel to the wall via two angles. Both sides of the table and the cable duct cover were calibrated for centimetre accurate positioning of the microphone. The recording of the microphone positions serves the subsequent visualization and evaluation of the results and the generation of the labels within the framework of SL. During test execution, the microphone cable is mounted inside the cable duct with the IoT-Kit Octopus attached to the underside of the table in order to avoid possible interference of the RIR by obstacles.

The centimeterwise distribution of the RIR recording over the locating range results in the amount of $l_{\text{pos}} = 1200$ different positions (100 per 10 cm $\times$ 10 cm square, see Figure 3). At each position, three recordings are taken to obtain sufficient training and test data. Due to physical conditions, hardware characteristics of the devices used, and noise, the three recordings at the same position are not identical, but have a similar profile. We use the first instance of each position as training data. In order to simplify processing and guarantee or improve accuracy, all recordings were taken in rows along the 60 cm long x-direction (along the cable duct) row by row along the 20 cm y-direction depicted in Figure 3. This leads to 180 recordings each row, in the order depicted in Table 1.

This is repeated the first ten rows, representing the width of the locating range. From the eleventh row, the same procedure is repeated for labels 7-12, leading to a total of 3600 recordings. After the introduction of a room change by an obstacle, the changed RIR is recorded again three times at the same 1200 positions in the same order to generate the online test data of 3600 additional files in the same order.The order also plays a role when testing the algorithm, since the test data vectors are classified in this order.

1) SIGNAL GENERATION

The signal is generated by means of an audio file over an ordinary loudspeaker, which is driven by the peripheral computer. The audio file is created by a Matlab script. The lower cutoff frequency or Schroeder frequency $f_{\text{Schroeder}}$ is for a room with a volume of $V = l \cdot b \cdot h = 3\ \text{m} \cdot 6\ \text{m} \cdot 2.5\ \text{m} = 45\ \text{m}^3$ and a reverberation time $T_N \approx 0.2$ s approximately:

$$f_{\text{Schroeder}} = 2\ \text{kHz} \cdot \sqrt{\frac{T_N}{V}} = 2\ \text{kHz} \cdot \sqrt{\frac{0.2}{45}} = 133.33\ \text{Hz} \tag{1}$$

In a heuristic procedure, however, the frequency of 100 Hz was determined. After the user presses a button on the IoT-Kit, the data acquisition process begins. The IoT-Kit sends a Message Queuing Telemetry Transport Protocol (MQTT) message using topic *play* to the peripheral computer, which triggers playback of the audio file. Figure 4 illustrates the machine to machine (M2M) communication for data acquisition.

2) SIGNAL RECORDING

The generated audio file containing a 100 Hz sine sound of 1 ms duration is played by a distributed cyber-physical system (CPS) [14]. Since this requires a certain amount of transmission and processing time, the IoT-Kit waits a heuristically determined waiting time span until the recording starts. This waiting time ensures synchronization between recording and playback of the signal. The recording time $t_r = 1$ second with a sampling rate of $f_s = 10$ kHz, resulting in the length of the raw data $l_{\text{raw}} = 10000$ measured values. We store all sampled measurements $s(mT)$ in a vector

$$\mathbf{v} = [s(T), s(2T), \ldots, s(l_{\text{raw}}T)]^T \tag{2}$$

which represents the raw data of a recording [4].

3) DATA TRANSMISSION

After recording, the data is sent to the cloud for storage and further processing via MQTT topic *transmit*. Due to limited storage ressources on the IoT-Kit, we store the measured values in a Byte array, where each value is represented by 2 Bytes. Regarding the maximum MQTT payload size of 1 kByte each message, we split the 20000 Bytes into 20 MQTT packages to be transmitted. Function *byte_to_int* calculates integer values from the submitted bytes. This allows us to leverage the maximum capacity of our IoT-Kit by outsourcing complex calculations and data structures.

4) DATA PREPROCESSING

After the data transfer is complete, the IoT-Kit sends a command in topic *convert* to trigger the *create.csv* node that runs a Python script to merge the 20 generated plain text files into a valid.csv file to convert the received data into a suitable format for further processing. To limit the amount of data, a Matlab script cuts 0.2 seconds from the recording of the pulse to create the work data from the raw data. In order to cut the most informative 0.2 s, we search for the highest peak, and cut shortly before that to extract the following $l_{\text{work}} = 2000$ values of the recording. After standardisation of these values, they serve as the work data $\mathbf{x}$ and basis for the

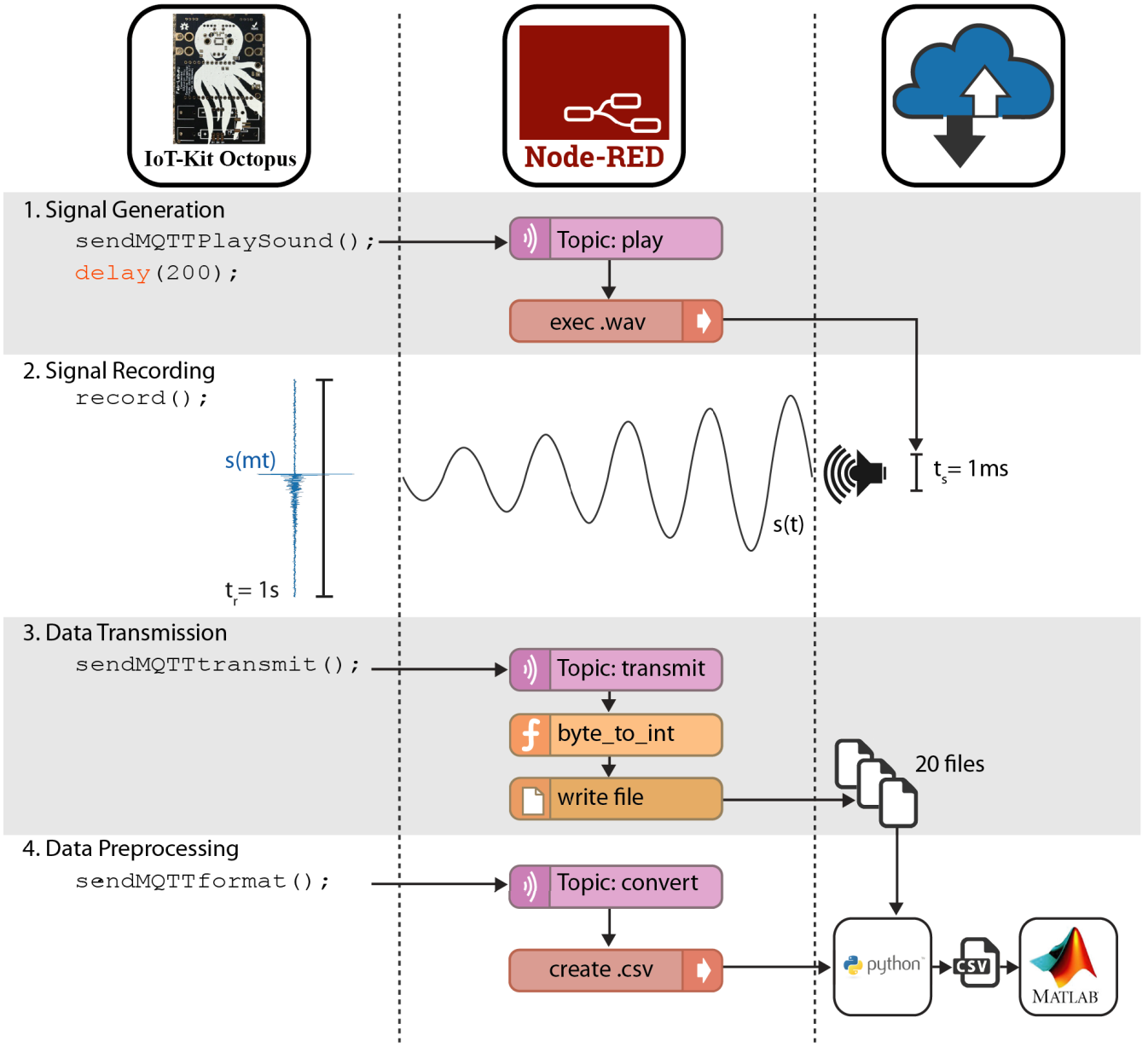

FIGURE 4. M2M via MQTT, after triggering the speaker (1) of the peripheral device, the sound is recorded (2) by the mobile device. The transmission (3) will be preprocessed and evaluated (4) to locate the microphone.

formation of the long-term memory (LTM). We formalize a work data vector $\mathbf{x}_n$ by:

$$\mathbf{x}_n = [x_{n;1}, \ldots, x_{n;l_{\text{work}}}] \tag{3}$$

where $n = 1, \ldots, 3600$ is the sequential number of the recording and entries of x_n are in the range of $[-1; 1]$. We formalize the training data set $\mathbf{M}$ as a matrix of column-wise arranged work data vectors x_n by

$$\mathbf{M} = \begin{pmatrix} x_{1;1} & x_{4;1} & x_{7;1} & \cdots & x_{3598;1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{1;l_{\text{work}}} & x_{4;l_{\text{work}}} & x_{7;l_{\text{work}}} & \cdots & x_{3598;l_{\text{work}}} \end{pmatrix} \tag{4}$$

and the *Offline Setup Test Data* as matrix **OffData1** starting with $x_{2;1}$, and **OffData2** starting with $x_{3;1}$ using the second, respectively third instance of the work data from the three recordings at each position. This way, we build a similar, not equal test set simulating the offline scenario, when no room change is applied. The *Online Setup Test Data*, simulating the online case of a changed room setup by an obstacle in the locating range (see the book in Figure 3), **OnData1**, **OnData2** and **OnData3** are structured in the same way. Since no training data is required as the algorithm learns while testing, we can use all three instances for testing purposes. The five test data sets will be handed over to the OOLA via

$\mathbf{O}^{\mathcal{L}}$, which incrementally transfers each single test data vector one after another to the classification algorithm.

C. ALGORITHMS

In this subsection, we describe the basic algorithms used and simulate the behaviour of LVQ.

1) K-Means

Since we work with limited resources, we first calculate twelve representative prototypes from each 100 training records per area using *K*-Means [15], resulting in the amount of $l_{\text{pro}} = 144$ prototypes for the $L = 12$ areas. *K*-Means splits the passed set of 100 data vectors in twelve groups also known as *Cluster* and returns their centers which we use for further classification. The calculation of clusters leads

Algorithm: *K*-Means

Input: Matrix $\mathbf{M}$; number of Clusters k

Choose k random data vectors from $\mathbf{M}$ as initial *new* prototypes, set *old* prototypes as empty matrix
while *new prototypes are not old prototypes* **do**
  for every point in $\mathbf{M}$ find the closest prototype
  Set *new* prototypes as the centroids of those assigned to the k *old* prototypes
end

Output: k prototypes of $\mathbf{M}$

to a small loss of information, a so-called in-sample error, which we accept in order to increase computational efficiency [5]. The in-sample error describes error of the model based on the training data. We reach classification rates of about 98 % (2 % in-sample error) in our example, which is a fair trade-off for minimizing the model from 1200 instances to 144. Cross-validation, by using the second instance of the three recordings at each position as training data set, leads to similar results. In this paper, our results refer to the use of the first instance of the three recordings at each position as training data.

2) KNN

The 144 *K*-Means generated representative prototypes are labeled according to their twelve origin areas. They have the same structure as the work data $\mathbf{x}_n$, renamed to $\mathbf{p}_n$:

$$\mathbf{p}_n = [p_{n;1}, \ldots, p_{n;l_{\text{work}}}] \tag{5}$$

where $n = 1, \ldots, 144$ and entries of p_n are in the range of $[-1; 1]$ due to standardization. However, they are assigned to an area by means of a label $l_n \in \{1, \ldots, L\} = \mathcal{L}$ and thus form the model of a *K*NN-algorithm [16], the LTM $\mathbf{C}^{\mathcal{L}}_{K\text{-Means}}$, given by:

$$\mathbf{C}^{\mathcal{L}}_{K\text{-Means}} = \begin{pmatrix} p^{\mathbf{C}}_{1;1} & \cdots & p^{\mathbf{C}}_{12;1} & p^{\mathbf{C}}_{13;1} & \cdots & p^{\mathbf{C}}_{144;1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ p^{\mathbf{C}}_{1;,l_{\text{work}}} & \cdots & p^{\mathbf{C}}_{12;l_{\text{work}}} & p^{\mathbf{C}}_{13;l_{\text{work}}} & \cdots & p^{\mathbf{C}}_{144;l_{\text{work}}} \\ 1 & \ldots & 1 & 2 & \ldots & 12 \end{pmatrix}, \tag{6}$$

is formally represented as follows:

$$\mathbf{C}^{\mathcal{L}}_{K\text{-Means}} = \begin{pmatrix} \mathbf{p}^{\mathbf{C}}_1 & \cdots & \mathbf{p}^{\mathbf{C}}_{144} \\ 1 & \ldots & 12 \end{pmatrix}, \tag{7}$$

where the superscript C inidicates the **C**onstant, unchanged LTM. The LTM is used to decide, which of the prototypes, respectively label or area, is most similar to the test data to classify and locate the microphone in a special area. Therefore, the SL algorithm compares a new signal with the set of all labeled prototypes and determines the K next records based on a selected criterion, e.g. the *Euclidean Distance*. *K*NN checks the classes (labels) of those K determined

Algorithm: *K*NN

Input: Labeled matrix $\mathbf{C}^{\mathcal{L}}$; new sample $\mathbf{x}_0$

Find the K nearest neighbors to $\mathbf{x}_0$ from $\mathbf{C}^{\mathcal{L}}$
Identify the class with the majority of the K points

Output: Class for sample $\mathbf{x}_0$

nearest neighbors and assigns the most frequently occurring class to the signal to be classified to estimate its position. Since the possible classes are directly related to a defined position, the position of the data record is determinedWe already performed the combination of *K*-Means and *K*NN in [4], resulting in about 88 % correct classifications applied on a 60 cm × 60 cm field on a table divided in 16 areas of 15 cm × 15 cm. The measuring points were intuitively distributed in a star shape in the squares, which we verify to the nearest centimeter in this paper.

This method, also known as *lazy learner*, has advantages with regard to the complexity of the algorithm used and thus the runtime. *K*NN algorithms are also particularly suitable for SBI by evaluating the RIR,

- because they're able to directly classify by multiple groups,
- from the evaluation of neighbors, resistance to distant outliers arises,
- since their input is only the training data, a *K*NN can be adapted to new training objects by OL.

3) LVQ

The left part of Figure 5 visualizes the classification of a simulation with 400 two-dimensional, randomly generated dummy data vectors assigned to four groups. The small points represent the data sets, the large points represent calculated prototypes, crosses in the respective colors represent the classification of the data sets based on the model of the calculated prototypes. The calculation of the 12 representative prototypes of each group causes a lack of information which leads to misclassifications of the training data (in-sample failure). If only *K*-Means is used to calculate the prototypes, the classification of the training data results in 95.25 % correct predictions. Especially in the border areas (solid black lines) errors often occur, see orange circles. The right picture shows an incremental manipulation of the prototypes using

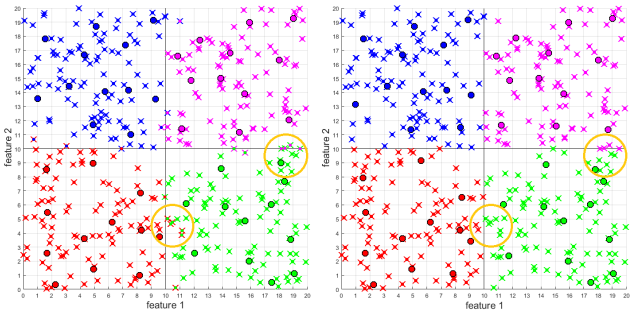

FIGURE 5. Improving *K*-Means with LVQ, the right plot right shows the moved representative prototypes and classification improvements.

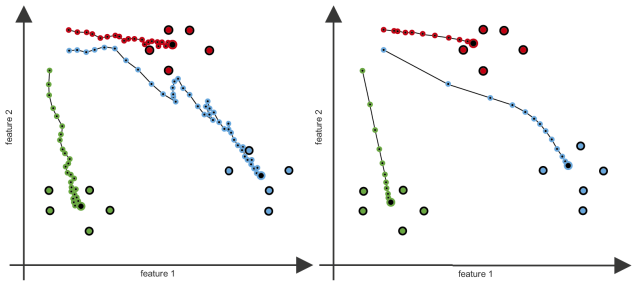

FIGURE 6. LVQ processed in online (left) and batch (right) mode [19].

LVQ leading to less faulty classifications. The prototypes were shifted in 25 incremental learning processes so that the classification is 100 % correct. The effect is particularly noticeable in the right center of the image, at the border between the purple and green areas. The LVQ [17] is another supervised classification method using a set of representative prototypes. First, the prototypes are initialized randomly. Then the nearest prototype is determined per data vector. If the class of the determined prototype corresponds to that of the data vector, the prototype is shifted in the direction of the current data vector. If the classes are different, the prototype is shifted in the opposite direction. The strength of the shift is controlled by a factor $\alpha(t)$, the so-called *learning rate*, which decreases depending on the time until all data vectors are processed or an abort criterion is fulfilled [18]. Figure 6 illustrates the process using the example of three randomly selected prototypes in the upper left corner and three sets of training vectors. The adjustment is done

- in the left picture by an incremental learning procedure with a learning rate of $\alpha = 0.05$,
- in the right picture by a batch learning procedure with a learning rate of $\alpha = 0.1$ [19].

In batch learning, the prototypes are updated only after all data vectors have been processed, in incremental learning, the prototypes are updated after each training pattern [19]. LVQ have already been extensively modified for the implementation of incremental learning methods [20]. One approach is to update the prototypes only if the influencing training pattern lies at one of the decision boundaries between the classes concerned. A basic LVQ in [21] defines

Algorithm: LVQ

Input: Labeled matrix $\mathbf{M}^{\mathcal{L}}$; new sample $\mathbf{x}_i$

Find the nearest neighbor to $\mathbf{x}_i$ from $\mathbf{M}^{\mathcal{L}}$
if *both belong to same class* **then**
  Shift this neighbor to $\mathbf{x}_i$
else
  Shift this neighbor away from $\mathbf{x}_i$
end

Output: $\mathbf{M}^{\mathcal{L}}$ with shifted data set

a factor s by

$$min\left(\frac{d_1}{d_2}, \frac{d_2}{d_1}\right) > s, \quad \text{where } d_1, d_2 \neq 0, \tag{8}$$

and controls in which cases an update takes place. LVQ 2.1 does not meet the convergence criterion, but provides a suitable entry point to make more complex extensions easier to understand. The distances d_1 and d_2 represent the length of the connection vectors to the nearest prototypes (Euclidean distance). The distance to the next prototype $\mathbf{p}_j$ of the same class as $\mathbf{x}_i$ is described by d_1. The distance d_2 represents the distance to the next prototype $\mathbf{p}_j$ of a class different from $\mathbf{x}_i$.

$$d_1 = |\mathbf{x}_i - \mathbf{p}_j| \quad \text{if } l_i = l_j, \tag{9}$$
$$d_2 = |\mathbf{x}_i - \mathbf{p}_j| \quad \text{if } l_i \neq l_j. \tag{10}$$

If d_1 and d_2 are almost equal, the minimum quotient is close to oneIf $\mathbf{x}_i$ is closer to one of the two prototypes $\mathbf{p}_1$ or $\mathbf{p}_2$, d_1 and d_2 are further apart and therefore $min(d_1/d_2, d_2/d_1)$ drops. The appropriate choice of s can be used to control which training patterns may trigger a learning process. Figure 7 illustrates the adjustment of the prototypes by LVQ 2.1. The prototypes $\mathbf{p}_1$ and $\mathbf{p}_2$ are represented by the colored crosses, red and blue dots mark the last 40 training patterns of the respective classes *red* and *blue*. The black dots and lines represent the positions and shifts of the prototypes during the

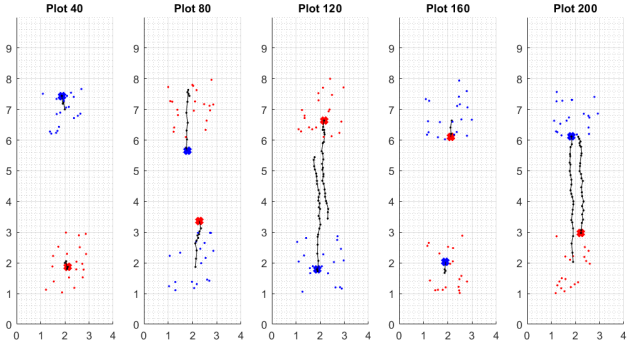

FIGURE 7. LVQ simulation with moving representative prototypes (abscissa: feature 1, ordinate: feature 2). When the test data sets switch their positions, the prototypes start moving. The effect can be undone by switching the test data positions again.

last 40 learning phases. The learning rate α is given by

$$\alpha = 0.05 + \frac{1}{100 + t} \tag{11}$$

where t stands for the actual number of updates. The time factor t is only increased if the prototypes are actually shifted, further $s = 0.5$ was selected.

Plot 40 of Figure 7 shows the position of the prototypes after the first 40 training patterns. The prototypes were initialized at coordinates (2,2) and (2,7), the training patterns were randomly generated with a maximum deviation of ± 1 around the initial positions of the prototypes. Since the training patterns of the red and blue class are always relatively close to their prototypes, weak shifts (black lines) of the prototypes rarely occur because $s < 0.5$ most of the time.

If the training patterns change their labels, as in Plot 80 and Plot 120 of Figure 7, the effect of the shift is more pronounced. The prototypes learn their new position from the labeled training patterns and forget the old one. So they adapt to the new conditions. By choosing s and α the speed of the effect can be controlled. If the labels are reversed again, as shown in Plot 160 and 200 of Figure 7, the effect can be undone. According to this principle, the effects of changes in space on the RIR can be learned.

In various other variants, the LVQ can be easily adapted to the requirements of the application, e.g. the simultaneous updating of several prototypes per training sample is possible.

A further extension which optimizes the displacement of the prototypes is the *Generalized Learning Vector Quantization* (GLVQ) developed by Sato and Yamada [21]. They show that approaches of e.g. Kohonen [17] give good results, but do not necessarily converge in the global optimum. By introducing a minimized cost function, the lowest possible error rate is achieved.

Finally, Sato and Yamada introduce the *Relative Distance* $\mu(\mathbf{x})$ with

$$\mu(\mathbf{x}) = \frac{d_1 - d_2}{d_1 + d_2} \tag{12}$$

where d_1 is the distance to the next prototype of the same class and d_2 the distance to the next prototype of another class. Since d_1 describes the distance to the prototype of the same class, $\mu(\mathbf{x})$ becomes negative if the classification is correct. To improve the error rate, $\mu(\mathbf{x})$ should decrease over the amount of data vectors entered similar to the factor s used in the LVQ 2.1 procedure. Sato and Yamada propose a function with a single maximum at $\mu = 0$, whose width decreases with increasing time to ensure that prototype updates are only performed at similar distances to the training pattern. The intensity of the update depends on μ, and thus on d_1 and d_2. By choosing μ appropriately, the convergence criterion can be met by this procedure.

D. SIMULATION RESULTS

While *K*-Means only calculates representative prototypes from the existing training data with the same label, it carries out a local optimization in the area. The shifting of the wrongly classified data sets of all labels by means of LVQ results in a global consideration of all prototypes. However, data sets in boundary areas represented by the *K*-Means prototypes may be closer to prototypes of neighboring areas, causing errors in classification. The LVQ adjustment shifts the prototypes of the boundary areas until all training data sets are correctly classified (in some cases only about 99 % could be reached) and thus obviously provides a sharper separation in the boundary areas (see Figure 5). In future tests of the OOLA, the success rates of classification by LVQ-manipulated prototypes $\mathbf{C}^{\mathcal{L}}_{\text{LVQ}}$, which correspond to the slightly shifted pure *K*-Means prototypes $\mathbf{C}^{\mathcal{L}}_{K\text{-Means}}$, will be compared. It is therefore examined if and how good these manipulations affect the classification rate on test data sets. To do this, a Matlab script adapts the previously selected set of prototypes via LVQ and compares the classification quotes. To investigate the stability of this effect, 60 sets of prototypes are manipulated using LVQ and the classification rates before and after are compared. The quotas are calculated by the ratio of the amount of correct predictions divided by total predictions. After evaluating the results, the following values can be determined:

- maximum quota $q_{\text{max}} = 92.3$ %
- minimum quota $q_{\text{min}} = 89.2$ %
- maximum improvement $i_{\text{max}} = 4.6$ %
- minimal improvement $i_{\text{min}} = 0.8$ %
- mean of the improvement $i_{\text{mean}} = 2.5$ %

The LVQ thus offers an opportunity to improve the *K*-Means prototypes, which achieve positive results in all tested cases with the same room conditions. On average about 2.5 % more data points can be classified correctly using LVQ-manipulated prototypes $\mathbf{C}^{\mathcal{L}}_{\text{LVQ}}$.

III. CASE STUDY - ONLINE OFFLINE LEARNING

In this section we first explain the components of the learning architecture and how they work together until we present our case study which helps understand the learning methodology.

A. ONLINE OFFLINE MEMORY

Even with improved prototypes, we still face the problem of environmental change. When applying our *Online Setup Test Data* (blue, red and yellow lines) from a changed environment to this model, the classification rates drop sharply, as shown in Figure 8. The colored lines show the ratio of so far correctly classified to so far processed test data vectors. We still achieve about 90 % correct classifications on our fingerprinting environment data (*Offline Setup Test Data*, green and purple line), but the classification rates of the LTM for online data are too poor to be used in a real, changing environment.

Fischer *et al.* [8] emphasize the requirements of modern applications in the field of autonomous robots or driving systems for the use of incremental learning methods. They refer to the complexity of human learning strategies, which, despite the use of incremental methods, preserve basic concepts or basic knowledge, but can nevertheless

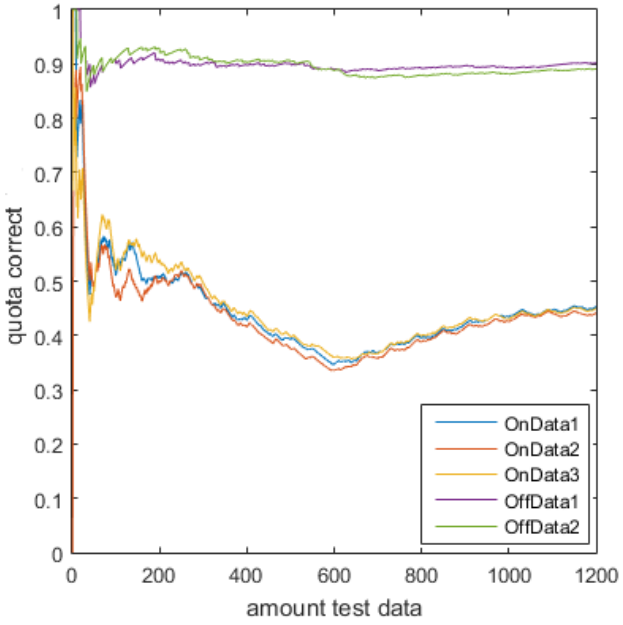

FIGURE 8. Classification quota of offline and online setup test data by LTM, showing bad results for changed environments. The x-axis represents the $l_{\text{pos}} = 1200$ work data vectors defined in Section II-B, the y-axis shows the ratio of so far correctly classified to so far processed test data vectors.

quickly adapt to important changes. They raise the question of how different mechanisms can be efficiently combined to ensure stability and flexibility at the same time, which are contradictory in their requirements. They address the so-called *Catastrophic Forgetting Effect* (CFE) [22], which occurs in incremental learning methods and describes the forgetting of already learned knowledge through incrementally acquired information. Another typical problem of incremental methods, in which the concept of classification (the relationship between input and output) changes, is also dealt with. The so-called *concept-drift* is avoided by the extension of a hybrid architecture, which consists of three components [8]:

1) A kind of static LTM to avoid forgetting and thus ensure stability. This classifier is generated by a batch learning process at the beginning of runtime and is not subject to any changes.
2) A flexible short-term memory (STM) as a second classifier, that starts without knowledge and adapts permanently and flexibly to the circumstances through incremental learning in order to respond to conceptual changes.
3) A decision element which compares the results of the classifications, in order to control the output of the OOLA and the training of the STM based on this data.

The essential function of the LTM is to counteract the CFE. By separating the batch process from the incremental process, it can be ensured that changes to the prototypes through OL do not affect the static knowledge. The static knowledge $\mathbf{C}^{\mathcal{L}}$ is generated before runtime in the described fingerprinting process (Section II-C1 to II-D).

The rating of the prototypes in the STM and the selection of the decision element, which of the two classifiers is responsible in the respective case, is made by evaluating the *Relative Similarity* introduced in [23]. They negate $\mu(\mathbf{x})$ from GLVQ to the relative similarity relSim($\mathbf{x}$):

$$\text{relSim}(\mathbf{x}) := -\mu(\mathbf{x}) = -\frac{d_1 - d_2}{d_1 + d_2} \tag{13}$$

This formally expresses the quality of the classification. The values of the relative similarity are rational numbers in the range of $[-1, 1] \ni \text{relSim}(\mathbf{x})$,

- Values of one indicate a high level of security for the classification, since the distance (d_1) to the prototype of the class corresponding to the label of $\mathbf{x}$ is small.
- values close to zero indicate a high degree of uncertainty, since the distance to the next correct prototype is approximately as large as the distance to the next incorrect prototype.
- Values at minus one indicate a wrong classification.

The latter two are of special importance for online learning. Since it makes no sense to learn already correctly classified vectors, the misclassified ones are especially interesting. They provide the highest gain in information. To test the OOLA we hand over the five test matrices

1) **OnData1**,
2) **OnData2**,
3) **OnData3**,
4) **OffData1**,
5) and **OffData2**

together with their origin labels described in Section II-B as online learning test matrix $\mathbf{O}^{\mathcal{L}}$, defined as:

$$\mathbf{O}^{\mathcal{L}} = \begin{pmatrix} \mathbf{x}_1^O & \cdots & \mathbf{x}_{l_{\text{pos}}}^O \\ l_1^O & \cdots & l_{l_{\text{pos}}}^O \end{pmatrix}, \tag{14}$$

one after another. A test data vector $\mathbf{x}_n^O$ equals the recordings $\mathbf{x}_n = [x_{n;1}, \ldots, x_{n;l_{\text{work}}}]$, the label $l_n^O \in \{1, \ldots, L\}$, with $n = 1, \ldots, l_{\text{pos}}$. These are classified by the LTM, which leads to multiple misclassifications, but improves plasticity by using the misclassified data vectors with their correct label (provided by SL) as additional prototypes in the STM [24]. After building the STM, only prototypes that cannot be correctly classified by both classifiers will be learned. Due to limited resources, shortest possible computing time and the adaptability of the algorithm, prototypes should also be removed from the STM when they are no longer useful.

B. LEARNING AND FORGETTING

To evaluate their usefulness, the value of the relative similarity is summed up per prototype in the STM, if it falls below a certain value (here 0), the prototype is often responsible for wrong classifications and should be removed from the

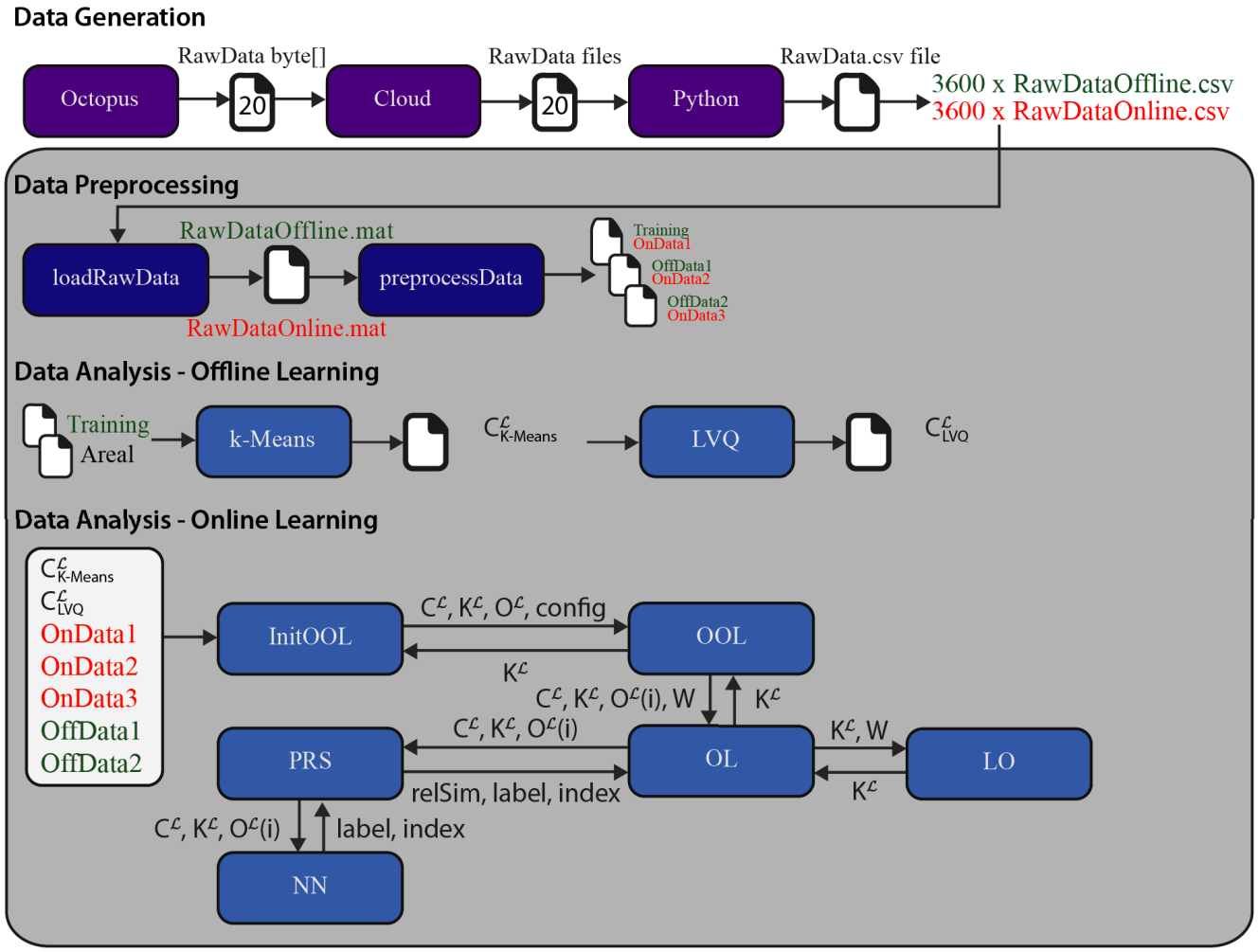

FIGURE 9. Overview of the overall process, after data generation and preprocessing for training and test data creation, the LTMs are built up first. These components are then used to test the OOLA.

STM [25]. In particular, the STM $\mathbf{K}^{\mathcal{L}}$ starts without any knowledge, i.e. $\mathbf{K}^{\mathcal{L}} = \emptyset$, two mechanisms are used for the generation of the incremental learned online knowledge:

1) **add prototypes:**
Based on the idea of [26], a *Workbuffer* $\mathbf{W}$ contains incorrectly classified online test data vectors $\mathbf{x}_i^O$ as *candidates* for a *promotion* to a STM prototype, up to a maximum memory capacity e. They are stored together with their correct label l_i^O and the relative similarity value $\text{relSim}^C(\mathbf{x}_i^O)$ calculated under usage of the prototypes of the LTM $\mathbf{C}^{\mathcal{L}}$ during classification. If the buffer is full, the *candidate* with the smallest value of relative similarity for each class occuring in $\mathbf{W}$ is stored in the STM $\mathbf{K}^{\mathcal{L}}$ as new prototype. This results in selecting data sets that have a high uncertainty and therefore are potentially useful.
2) **remove prototypes:**
Similar to [25] the usefulness of prototypes is evaluated by their relative similarity. For this purpose each STM prototype is assigned aparameter relSim^K for the accumulation of the relative similarities, which describes its usefulness for cost minimization. If a prototype represents the best matching unit (the nearest neighbor) for a new data vector, the evaluated relative similarity is added to this parameter. After a heuristically determined number of m learned data vectors, all prototypes with negative accumulated relative similarity are deleted.

The effects on the error rate of the classification are examined, regarding the number of processed data vectors m until deletion, and the variable size e of the buffer $\mathbf{W}$. Workbuffer $\mathbf{W}$ is defined by:

$$\mathbf{W} = \begin{pmatrix} \mathbf{x}_1^W & \dots & \mathbf{x}_e^W \\ l_1^W & \dots & l_e^W \\ \text{relSim}_1^W & \dots & \text{relSim}_e^W \end{pmatrix}, \tag{15}$$

where a *candidate* equals a column of $\mathbf{W}$ including:

- the vector $\mathbf{x}_n^W$, which equals the i-th processed online test data vector $\mathbf{O}^{\mathcal{L}}(i) = \mathbf{x}_i^O = [x_{i;1}^O, \dots, x_{i;l_{\text{work}}}^O]$ of the online test data matrix $\mathbf{O}^{\mathcal{L}}$, which lead to the n'th misclassification and therefore a negative $\text{relSim}^C(\mathbf{x}_i^O)$,

- the label $l_n^W \in \{1,\ldots,L\} = \mathcal{L}$ is the label l_i^O of this processed test data vector,
- and relSim_n^W is the (negative) value of the $\text{relSim}^C(\mathbf{x}_i^O)$ calculated by the usage of the LTM $\mathbf{C}^{\mathcal{L}}$, which leads to $\text{relSim}_n^W \in [-1;0[$ due to selection strategy of the decision element, just adding prototypes with negative values of $\text{relSim}(\mathbf{x}_i^O)$ as candidates,

with $n \in \{1,\ldots,e\}$. The negative value serves as indicator of the quality of test data vector $\mathbf{O}^{\mathcal{L}}(i)$ to be added as candidate with its origin label l_i^O (given by the SL) and the negative value $\text{relSim}(\mathbf{x}_i^O)$ to $\mathbf{W}$. If e is reached, for each existing label l_n^W in $\mathbf{W}$, the one of the candidates carrying this label with the lowest relSim^W will be *promoted* to a representative prototype $\mathbf{p}_{l_{\text{act}}+1}^K$ and therefore be added to the STM $\mathbf{K}^{\mathcal{L}}$, given by:

$$\mathbf{K}^{\mathcal{L}} = \begin{pmatrix} \mathbf{p}_1^K & \cdots & \mathbf{p}_{l_{\text{act}}}^K \\ l_1^K & \cdots & l_{l_{\text{act}}}^K \\ \text{relSim}_1^K & \cdots & \text{relSim}_{l_{\text{act}}}^K \end{pmatrix}, \tag{16}$$

where l_{act} gives the amount of prototypes actually in $\mathbf{K}^{\mathcal{L}}$. The added prototype $\mathbf{p}_{l_{\text{act}}+1}^K$ consists of the following elements:

- the prototype $\mathbf{p}_{l_{\text{act}}+1}^K$ equals the data vector $\mathbf{x}_n^W$ of the candidate $\mathbf{W}(n)$ which became promoted to the prototype,
- the label $l_{l_{\text{act}}+1}^K \in \{1,\ldots,L\} = \mathcal{L}$ is the label l_n^W of this promoted candidate,
- and $\text{relSim}_{l_{\text{act}}+1}^K$ is the accumulated value added up over all classification usages of this prototype,

with the initial value of the accumulated Relative Similarity $\text{relSim}_n^K = 0$. Fischer et al. conclude in [8], that the appropriate choice of the parameters depends on the use-case and the complexity and accuracy of the resulting models. The STM in the context of this work has the task of learning changes in space, leading to a modified RIR. The test data vectors $\mathbf{O}^{\mathcal{L}}(i)$ are classified by both classifiers $\mathbf{C}^{\mathcal{L}}$ and $\mathbf{K}^{\mathcal{L}}$ in the hybrid OOLA. LTM $\mathbf{C}^{\mathcal{L}}$ and STM $\mathbf{K}^{\mathcal{L}}$ determine the values relative similarities $\text{relSim}_{\text{temp}}^C(\mathbf{x}_i^O)$ and $\text{relSim}_{\text{temp}}^K(\mathbf{x}_i^O)$ and label l_i^C and l_i^K according to their prototypes. These values are transferred to a decision algorithm, deciding

- which classification l_{Out} will be output after evaluating the relative similarities, choosing the label of the classifier with the highest $\text{relSim}_{\text{temp}}$ by,

$$l_{\text{Out}} = \begin{cases} l_i^C & \text{if } \text{relSim}_{\text{temp}}^C(\mathbf{x}_i^O) \geq \text{relSim}_{\text{temp}}^K(\mathbf{x}_i^O) \\ l_i^K & \text{else} \end{cases} \tag{17}$$

- and whether the data vector is used to train the STM. We add $\mathbf{O}^{\mathcal{L}}(i)$ to $\mathbf{W}$ if
  - the LZG $\mathbf{C}^{\mathcal{L}}$ leaves a better result than the KZG $\mathbf{K}^{\mathcal{L}}$

$$\text{relSim}_{\text{temp}}^K(\mathbf{x}_i^O) < \text{relSim}_{\text{temp}}^C(\mathbf{x}_i^O) < 0, \tag{18}$$

    but both classifiers still fail,
  - or if the LTM classification is worse than the classification of the STM,

$$\text{relSim}_{\text{temp}}^C(\mathbf{x}_i^O) < \text{relSim}_{\text{temp}}^K(\mathbf{x}_i^O) \tag{19}$$

TABLE 2. Relative similarities, classification decisions and their reasons - plot 1 of figure 10.

Data	relSim^C	relSim^K	l_i^C	l_i^K	l_{Out}
$\mathbf{p}^K(1)$	0.89	–	1	–	1
$\mathbf{p}^K(2)$	0.89	–	2	–	2
$\mathbf{x}_1^O$	0.20	0.22	1	1	1
$\mathbf{x}_2^O$	−0.20	−0.22	2	2	2
$\mathbf{x}_3^O$	−0.60	−0.64	2	2	2

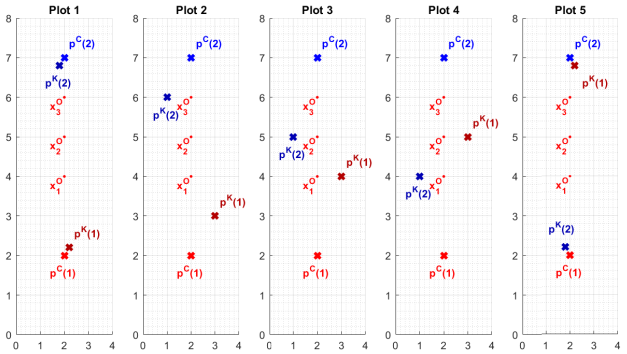

FIGURE 10. Effects of the existing prototypes for the *Relative Similarity* and their effects on the selection of online learning prototypes $\mathbf{p}^K$ regarding just two areas (abscissa: feature 1, ordinate: feature 2). The light red ($\mathbf{p}^C(1)$) and light blue ($\mathbf{p}^C(2)$) crosses show the LTM $\mathbf{C}^{\mathcal{L}}$ prototypes of the both areas 1 and 2. Dark red ($\mathbf{p}^K(1)$) and dark blue ($\mathbf{p}^K(2)$) crosses show 5 hypotetic online learned prototypes. The light red points $\mathbf{x}_1^O - \mathbf{x}_3^O$ are examples of test data vectors from area 1 (SL) to be classified. Table 2 - Table 6 show the values of the different Relative Similarities and decisions of the OOLA.

    which generates many candidates with little chance to get promoted,
  - or if both classifiers leave no better result than 0.2

$$\max\left(\text{relSim}_{\text{temp}}^C(\mathbf{x}_i^O), \text{relSim}_{\text{temp}}^K(\mathbf{x}_i^O)\right) < 0.2 \tag{20}$$

    to cover the areas of high uncertainity.

Due to limited resources and computational time issues, we limit the amount of prototypes for each label to 13, resulting in $l_{\text{proOn}} = 156$ online prototypes $\mathbf{p}_n^K$, where $n = 1,\ldots,l_{\text{proOn}}$. If a prototype of a label is to be learned that already has the maximum number of prototypes, the prototype with the actually lowest accumulated relSim^K is replaced by the new prototype.

C. CLOSER LOOK AT THE RELATIVE SIMILARITY

The *Relative Similaity* plays the main role for the generation of the STM $\mathbf{K}^{\mathcal{L}}$. It is responsible for evaluating the value of a test data vector to be stored in the Workbuffer $\mathbf{W}$ and is the criteria for those candidates to be promoted to an OL vector $\mathbf{p}^K$. As already mentioned, it makes no sense for the STM to learn information already present in the LTM. Therefore, [8] recommends learning data vectors which provide a negative relSim^C. Figure 10 shows five different setups for potential OL vectors $\mathbf{p}^K$. In Plot 1 of Figure 10, the data vectors near the LTM $\mathbf{C}^{\mathcal{L}}$ have been promoted and learned, even having a high $\text{relSim}^C = 0.89$ as summarized in Table 2.

TABLE 3. Relative similarities, classification decisions and their reasons - plot 2 of figure 10.

Data	relSim^C	relSim^K	l_i^C	l_i^K	l_{Out}
$\mathbf{p}^K(1)$	0.49	–	1	–	1
$\mathbf{p}^K(2)$	0.49	–	2	–	2
$\mathbf{x}_1^O$	0.20	0.23	1	1	1
$\mathbf{x}_2^O$	−0.20	−0.23	2	2	2
$\mathbf{x}_3^O$	−0.60	−0.52	2	2	2

The classifications of the test data vectors $\mathbf{x}_1^O - \mathbf{x}_3^O$ from area 1 (with label 1) can be interpreted as follows:

- The test data vector $\mathbf{x}_1^O$ is classified correctly (green colored font in field l_i^C) by the STM $\mathbf{K}^{\mathcal{L}}$, but the LTM would have lead to the same result anyhow, so the prototype was kind of useless.
- The test data vectors $\mathbf{x}_2^O$ and $\mathbf{x}_3^O$ are classified wrong (red colored font in field l_i^C), the STM can not provide a better relSim^K.

It can be said that learning test data vectors with high relative similarities does not have a particularly beneficial effect.

Comparing the results of Table 3 confirms the observation, however, as the distance between the prototypes learnt ($\mathbf{p}^K(1)$ and $\mathbf{p}^K(2)$) and the LTM prototypes ($\mathbf{p}^C(1)$ and $\mathbf{p}^C(2)$) increases, the relative similarities improve slightly.

Table 4 referring Plot 3 of Figure 10 shows no new insights, but confirms the improvement of the relative similarities especially for $\mathbf{x}_3^O$.

Finally, Table 5 related to Plot 4 of Figure 10 shows the intended effect, the learnt test data vectors which have been classified wrong by the LTM lead to correct classifications for $\mathbf{x}_2^O$ and $\mathbf{x}_3^O$.

Additionally, Table 6 related to Plot 5 of Figure 10 confirms the results. The learnt test data vectors which have been classified wrong by the LTM lead to correct classifications for $\mathbf{x}_2^O$ and $\mathbf{x}_3^O$.

It can also be stated that the relSim^C values for $\mathbf{x}_1^O - \mathbf{x}_3^O$ remain constant, while the relSim^K values improve when using STM prototypes with negative relSim^C. However, this is only happening if those test data vectors have not already been correctly classified by the LTM (see $\mathbf{x}_1^O$). Selecting the highest *Relative Similarity* according to equation (17) leads to improvement of the overall number of correct classifications.

TABLE 4. Relative similarities, classification decisions and their reasons - plot 3 of figure 10.

Data	relSim^C	relSim^K	l_i^C	l_i^K	l_{Out}
$\mathbf{p}^K(1)$	0.17	–	1	–	1
$\mathbf{p}^K(2)$	0.17	–	2	–	2
$\mathbf{x}_1^O$	0.20	0.17	1	1	1
$\mathbf{x}_2^O$	−0.20	−0.17	2	2	2
$\mathbf{x}_3^O$	−0.60	−0.23	2	2	2

TABLE 5. Relative similarities, classification decisions and their reasons - plot 4 of figure 10.

Data	relSim^C	relSim^K	l_i^C	l_i^K	l_{Out}
$\mathbf{p}^K(1)$	−0.17	–	1	–	2
$\mathbf{p}^K(2)$	−0.17	–	2	–	1
$\mathbf{x}_1^O$	0.20	−0.17	1	1	1
$\mathbf{x}_2^O$	−0.20	0.17	2	2	1
$\mathbf{x}_3^O$	−0.60	0.23	2	2	1

TABLE 6. Relative similarities, classification decisions and their reasons - plot 5 of figure 10.

Data	relSim^C	relSim^K	l_i^C	l_i^K	l_{Out}
$\mathbf{p}^K(1)$	−0.89	–	1	–	2
$\mathbf{p}^K(2)$	−0.89	–	2	–	1
$\mathbf{x}_1^O$	0.20	−0.22	1	1	1
$\mathbf{x}_2^O$	−0.20	0.22	2	2	1
$\mathbf{x}_3^O$	−0.60	0.64	2	2	1

D. CASE STUDY

Figure 9 shows the overall process with the components involved. As part of *Data Generation*, 7200 individual.csv files are generated, which represent RIR recordings. During *Data Preprocessing*, these recordings are processed to the training data vectors in **M** and the offline test data vectors in **OffData1** and **OffData2**, respectively the online test data vectors in **OnData1**, **OnData2** and **OnData3** with their associated labels. The first part of the learning section, the Section *Data Analysis - Offline Learning* describes the generation of both LTM the *K*-Means prototypes $\mathbf{C}_{K\text{-Means}}^{\mathcal{L}}$, which are trained offline in batch mode by *K*-Means, and the incrementally improved LVQ-manipulated prototypes $\mathbf{C}_{\text{LVQ}}^{\mathcal{L}}$ to determine the suitability of LVQ for improving the classification quality of the LTM. The second part of the learning section, the Section *Data Analysis - Online Learning* describes the OOLA, which is adapting to changes by evaluation of the STM $\mathbf{K}^{\mathcal{L}}$ without losing existing knowledge in the LTM.

In order to compare the quality of the STM and to investigate the behaviour of the algorithms, the Matlab script `InitOOL.m` loads the prepared data sets and calls the main program `OOL.m` five times with different work data. This is repeated in five different test cases C1-C5:

C1) *Limitation of the number of prototypes to* 13*:*
Each area (label) has a maximum of 13 prototypes $\mathbf{p}^K$ in the STM $\mathbf{K}^{\mathcal{L}}$.

C2) *Verification with 19 prototypes per label:*
Increasing the maximum amount to 19 prototypes $\mathbf{p}^K$ each label in the STM $\mathbf{K}^{\mathcal{L}}$ to determine the trade-off between improvement increased calculating time.

C3) *Random order of test data:*
Using 13 prototypes $\mathbf{p}^K$ each area and bringing in the test data vectors randomly instead of using the data generation order described in Section II-B.

C4) *Further criterion for the deletion of prototypes:* Using 13 prototypes $\mathbf{p}^K$ each area determining, if the deletion of the prototype, which was not involved in a positive classification for the longest time leads to better results than using the one with the lowest accumulated relative similarity relSim^K.

C5) *Only learning with wrong LTM classification:* Using 13 prototypes $\mathbf{p}^K$ each area disallowing to learn prototypes when the LTM provides a correct classification (only using the condition of Equation 18).

While processing these test data sets, the algorithm will fill the buffer $\mathbf{W}$ as described above, until it reaches its maximum size e. Each time an online data test set $\mathbf{O}^{\mathcal{L}}(i)$ is classified wrong by the LTM, it will be added to $\mathbf{W}$ as potential candidate. When the size of $\mathbf{W}$ reaches e, algorithm `LO.m` will add one candidate of each label existing in $\mathbf{W}$ as prototype to the STM $\mathbf{K}^{\mathcal{L}}$, together with the correct label, the origin of the online test data vector (provided by SL), and an initial value (we use 0) of the accumulated relative similarity $\text{relSim}_e^{\mathbf{W}}$. The candidate with the lowest $\text{relSim}_e^{\mathbf{W}}$ is selected and thus represents the highest gain in information. After learning m test data vectors, the STM prototypes with a negative $\text{relSim}_e^{\mathbf{W}}$ will be deleted. These sequences will be repeated until the 6000 test data vectors are processed in five groups, respectively the five online test data sets:

1) The LTM $\mathbf{C}^{\mathcal{L}}$ and an empty STM $\mathbf{K}^{\mathcal{L}}$ as well as the first online test data set **OnData1** as $\mathbf{O}^{\mathcal{L}}$ after bringing in the room change. The OOLA starts learning the room changes and returns a STM with selected prototypes. Compared to the blue line in Figure 8, the blue lines of Figure 11 perform better, depending on the values of e and m.
2) The LTM $\mathbf{C}^{\mathcal{L}}$ and the STM $\mathbf{K}^{\mathcal{L}}$ formed in step 1), as well as the second online test data set **OnData2** as $\mathbf{O}^{\mathcal{L}}$ after introducing the room change. The OOLA applies the learned knowledge and constantly expands it. The red lines of Figure 11 reach similar classification quotas.
3) The LTM $\mathbf{C}^{\mathcal{L}}$ and the STM $\mathbf{K}^{\mathcal{L}}$ expanded in step 2), as well as the third online test data set **OnData3** as $\mathbf{O}^{\mathcal{L}}$ after the introduction of the spatial change. The OOLA applies the learned knowledge and constantly expands it. The orange lines of Figure 11 verify the results of step 1 and 2.
4) The LTM $\mathbf{C}^{\mathcal{L}}$ and the STM $\mathbf{K}^{\mathcal{L}}$ extended in step 3) as well as the offline test data set **OffData1** as $\mathbf{O}^{\mathcal{L}}$ before introducing the room change. This step checks if the CFE can be avoided by the static LTM. The OOLA classifies according to the evaluation of the learned STM and the existing LTM. Erroneous classifications of the LTM are learned as new knowledge in the STM, as in the previous calls. Knowledge of the STM that is no longer required is replaced by relevant information. The purple lines of Figure 11 confirm the avoidance of the CFE.
5) The LTM $\mathbf{C}^{\mathcal{L}}$ and the STM $\mathbf{K}^{\mathcal{L}}$ expanded in step 4) as well as the second offline test data set **OffData2** as $\mathbf{O}^{\mathcal{L}}$ before introducing the room change. The OOLA classifies as in step 4 and adjusts itself continuously.

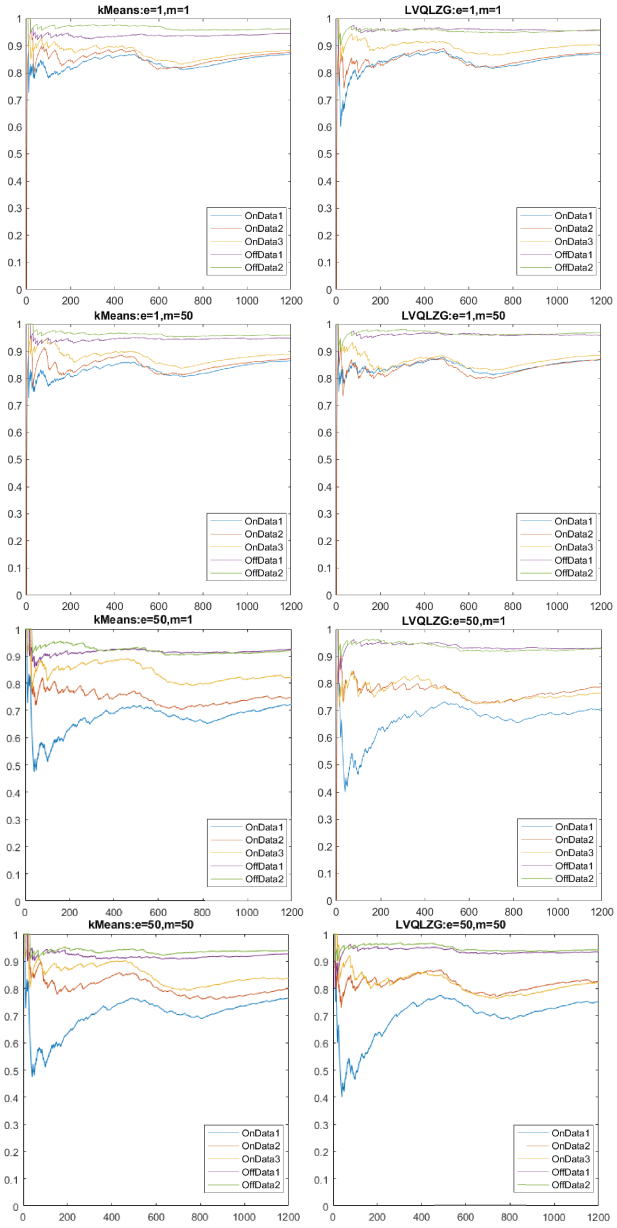

FIGURE 11. Classification quota of offline and online setup test data by OOLA, showing good results for changed environments and improved results for standard room setup, regarding various combinations of e and m. The x-axis represents the $l_{\text{pos}} = 1200$ work data vectors defined in Section II-B, the y-axis shows the ratio of so far correct classified to so far processed test data vectors.

The sequence of the calls is carried out with different parameters for the size e of the cache $\mathbf{W}$ and the number m of learning processes until the deletion of the prototypes from $\mathbf{K}^{\mathcal{L}}$. Furthermore, `InitOOL.m` collects statistics about the processes during the classification and their results. The full results and source codes are accessible via https://cosy.umwelt-campus.de/software. After processing the five test data sets, `InitOOL.m` erases the STM and restarts the process with the next combination of e and m using a new STM $\mathbf{K}^{\mathcal{L}} = \emptyset$.

For each of the five test data sets `InitOOL.m` calls the algorithm `OOL.m` with

- the actual LTM $\mathbf{C}^{\mathcal{L}}_{K\text{-Means}}$ or $\mathbf{C}^{\mathcal{L}}_{\text{LVQ}}$,
- the actual STM $\mathbf{K}^{\mathcal{L}}$
- the actual online test data set $\mathbf{O}^{\mathcal{L}}$,
- and the actual configuration of e and m

and is awaiting the return of STM $\mathbf{K}^{\mathcal{L}}$ after processing the online test data set $\mathbf{O}^{\mathcal{L}}$.

`OOL.m` initializes the Workbuffer $\mathbf{W}$ if necessary and represents the basic application to learn the changed RIR by passing algorithm `OL.m`

- the LTM,
- the STM,
- the individual online test data vectors $\mathbf{O}^{\mathcal{L}}(i)$ incrementally,
- and the Workbuffer $\mathbf{W}$

and is awaiting the return of the STM $\mathbf{K}^{\mathcal{L}}$ after the incremental processing of the individual online test data vector $\mathbf{O}^{\mathcal{L}}(i)$.

`OL.m` requests

- the index of the nearest prototypes of the same label as $\mathbf{O}^{\mathcal{L}}(i)$,
- the index of the nearest prototypes of a different label as $\mathbf{O}^{\mathcal{L}}(i)$,
- the relative similarity $\text{relSim}^{C}_{\text{temp}}$ of the LTM,
- and the relative similarity $\text{relSim}^{K}_{\text{temp}}$ of the STM, if the conditions are met (an existing prototype of the same class as $\mathbf{O}^{\mathcal{L}}(i)$ and an existing prototype of different class from $\mathbf{O}^{\mathcal{L}}(i)$ in the STM $\mathbf{K}^{\mathcal{L}}$)

from `PRS.m` and therefore passes the LTM, STM and online test data vector $\mathbf{O}^{\mathcal{L}}(i)$. `PRS.m` uses algorithm `NN.m`, representing a 1-nearest-neighbor classifier, to get these information.

`OL.m` further hands over the STM $\mathbf{K}^{\mathcal{L}}$ and the filled Workbuffer $\mathbf{W}$ to algorithm `LO.m`, if the conditions are met, which adapts the STM. Algorithm *Learn Online* (LO) is the main component for learning the modified RIR. Each time the amount of e records in the work buffer $\mathbf{W}$ is reached, they are passed to LO along with the current STM. LO determines the best candidate for each label present in $\mathbf{W}$ and adds it to the STM. After processing each existing label, all candidates are erased, $\mathbf{W} = \emptyset$. The e parameter can therefore affect the quality of the learned prototypes and the speed of growth of the STM. The higher e, the higher the quality of the learned prototypes should be, but the STM takes longer to fill. The lower e, the faster the Workbuffer gets learned, e.g. $e = 1$, each candidate is learned immediately. LO further checks the amount of incrementally processed test data vectors and starts the deletion of STM prototypes with negative accumulated relative similarity relSim^{K}. It should be emphasized once again that Algorithm `OOL.m` receives the test data individually, RIR for RIR one after the other, and always learns immediately when $\mathbf{W}$ reaches the size e. Due to faster simulation results and for reasons of organization of work processes we have not focused on a variety of agents to be located. Since the data is transferred RIR-wise, however, the procedure used approximates the workflow in a multi-agent scenario.

Algorithm: LO

Input: STM $\mathbf{K}^{\mathcal{L}}$; Workbuffer $\mathbf{W}$

for *each existing label in the Workbuffer* **do**
  Find the candidate with the lowest relative similarity
  if *STM for this label is not full* **then**
    Learn candidate
  else
    Replace existing prototype of the STM with lowest accumulated relative similarity
  end
  if *number of learning operations until deletion is reached* **then**
    delete all prototypes with negative accumulated relative similarity
  end
end

Output: new STM $\mathbf{K}^{\mathcal{L}}$; empty Workbuffer $\mathbf{W}$

TABLE 7. Exemplary view of errors using combinations of test set (S) and test case (C) for combinations of e and m (here $e = 1$ and $m = 1$ as in upper pictures of Figure 12 using pure K-Means LTM $\mathbf{C}^{\mathcal{L}}_{K\text{-Means}}$).

C: \ S:	OffData2	OffData1	OnData3	OnData2	OnData1
C5	56	71	122	145	150
C4	42	64	139	148	156
C3	109	95	164	185	227
C2	40	70	134	142	155
C1	47	66	140	148	156

IV. RESULTS

The overall results show an increase of the average classification rate to about 95 % for unchanged environments and the ability of the algorithms to adapt to changed environments by increasing from 72 % to 91 % (configuration dependent) of the OOLA compared to about 45 % (see Figure 8) in a pure offline setup with just LTM.

These results could even be improved, if not limiting the amount of prototypes each area, but this leads to very long processing times. For comparison, unlimited learning during first experiments led to 1198 prototypes at 6000 test data vectors, where the classification of the last 1200 vectors from **OffData2** required 3957.71 seconds, resulting in 37 classifications being incorrect (quota of about 96.92 %). Examples limited to 13 prototypes each label need about $100 - 150$ seconds, resulting in $40 - 50$ classifications being incorrect (quota of about $95.83 - 96.67$ %) at 6000 test data vectors to classify the last 1200 vectors from **OffData2** (overall $141 - 146$ prototypes).

The diagrams of Figure 11 show eight success rates of classification under different configurations of e and m with

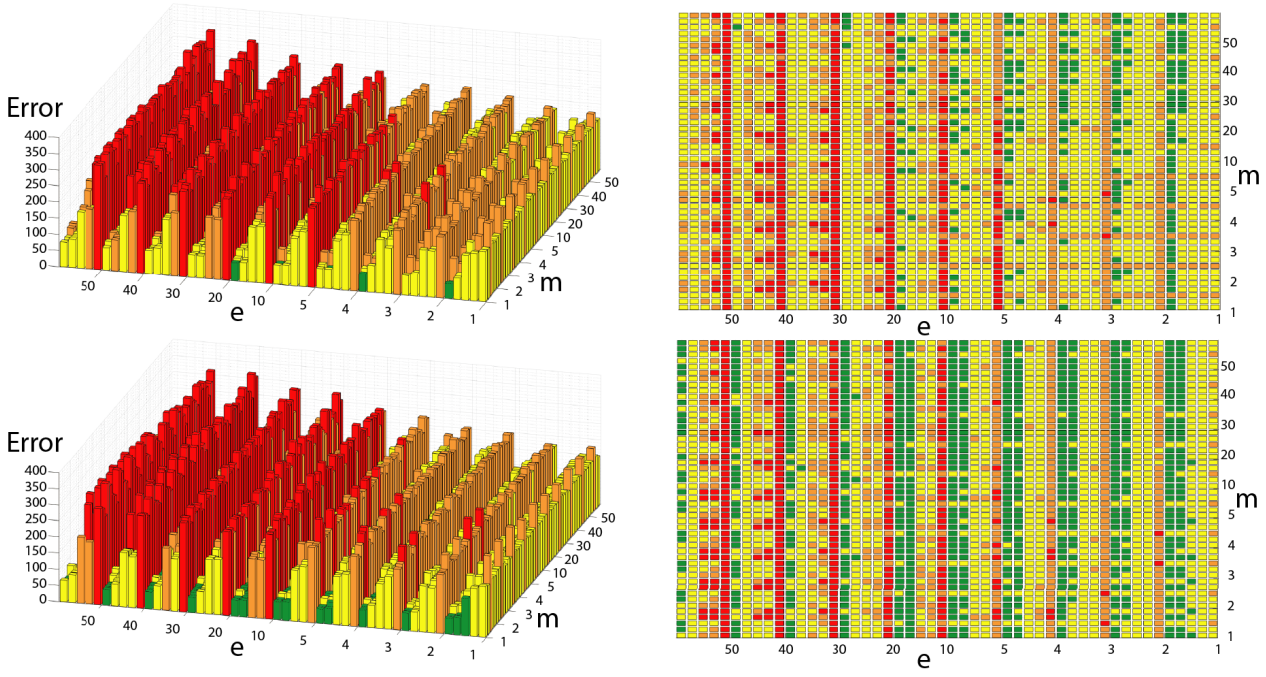

FIGURE 12. Total failures with and without LVQ depending on e, $m = 1, 2, 3, 4, 5, 10, 20, 30, 40, 50$.

TABLE 8. Color codes figure 12.

Test data	Color	Errors in %
Online	green	< 10
Online	yellow	< 15
Online	orange	< 20
Online	red	≥ 20
Offline	green	< 5
Offline	yellow	< 7.5
Offline	orange	< 10
Offline	red	≥ 10

standard and improved LTM. With e increasing from 1 to 50, the STM will not learn until e potential candidates for learning new prototypes are reached, which explains the sharp drop in the blue curve in the lower four plots, using $e = 50$. However, the effect should have a positive long-term effect with larger test data sets, as the quality of the prototypes should increase, due to selection of the best prototype each label is obsolete while $e = 1$. Because of the increase of m from 1 to 50, prototypes of the STM that have just been learned are no more deleted immediately, once they are responsible for a single wrong classification, and thus have a negative accumulated relative similarity. This can have a positive effect on the success rates, because they may start fitting in the long run. By processing the test data line by line, only test data in the $1-6$ range will be processed up to the 600-th data set. Starting from 601 the test data of the ranges $7-12$ are classified, the range change leads to a strong decrease of the quota, since no current prototypes of these ranges are available in the STM yet.

TABLE 9. Color codes figure 13.

Color	Error difference
green	> 60
light green	> 0
orange	≤ 0
red	< 60

The effect of the improved LTM is not visible in the plots, but a closer look at the classification rates shows that LVQ manipulation leads to an improvement in 1783 of 2500 cases. Figure 12 shows the errors of all five test runs bundled together. The upper two images show the errors when using the non-manipulated LTM, the lower ones the errors when using the LVQ-manipulated LTM. The left pictures show the errors as bar charts, each bar showing the classification error while processing the 1200 test data vectors of the test data set, the right pictures show a top view to improve the clear visibility of all results. The classification errors of the five test data sets **OnData1**, **OnData2**, **OnData3**, **OffData1** and **OffData2** are displayed repeatedly from right to left along the e-scale for each tested configuration of e and m. The five test cases C1-C5:

C1) Limitation of the number of prototypes to 13
C2) Verification with 19 prototypes per label
C3) Random order of test data
C4) Further criterion for the deletion of prototypes
C5) Only learning with wrong LTM classification

are repeatedly shown for each different combination of e and m in this order along the m-scale from bottom to top. So the 25 bars in each 5×5 square of a single combination of e and m

TABLE 10. Matrix of the sums of classification errors over the five different configurations using pure *K*-Means prototypes $\mathbf{c}^{\mathcal{L}}_{K\text{-Means}}$. Each field shows the amount of misclassifications per 30000 test data vectors (per row/column 300000, total 3000000). Green filled cells show lowest amount of errors in each column (red: highest). Green font indicates lowest failure amount each line (red: highest).

e: \ m:	1	2	3	4	5	10	20	30	40	50	$\sum$
1	2971	3002	2969	2947	2961	2860	2867	2824	2858	2837	39096
2	3241	3247	3250	3214	3189	3167	3013	3004	3008	3002	31335
3	3431	3397	3488	3426	3358	3327	3322	3205	3167	3198	33319
4	3512	3460	3536	3450	3431	3407	3350	3379	3315	3358	34198
5	3596	3633	3613	3666	3557	3529	3501	3390	3365	3328	35178
10	3887	3879	3733	3801	3795	3565	3714	3637	3546	3577	37134
20	3928	3928	3891	3838	3903	3878	3713	3664	3655	3698	38096
30	4189	4189	4189	4196	4125	4041	3950	3894	3896	3892	40561
40	4393	4393	4393	4412	4426	4360	4155	4109	4079	4061	42781
50	4546	4546	4546	4546	4463	4415	4382	4258	4258	4219	44179
$\sum$	37694	37674	37608	37496	37208	36549	35967	35364	35147	35170	365877

TABLE 11. Matrix of the sums of the classification errors using LVQ-manipulated LTM $\mathbf{c}^{\mathcal{L}}_{\text{LVQ}}$ over the five different configurations. Each field shows the amount of misclassifications per 30000 test data vectors (per row/column 300000, total 3000000). Green filled cells show lowest amount of errors in each column (red: highest). Green font indicates lowest failure amount each line (red: highest).

e: \ m:	1	2	3	4	5	10	20	30	40	50	$\sum$
1	2811	2826	2766	2753	2741	2699	2735	2719	2718	2738	27506
2	3051	2956	3035	3077	2957	2935	2910	2926	2946	2820	29613
3	3262	6234	3225	3132	3140	3157	3033	3074	3118	3082	31457
4	3334	3207	3250	3223	3243	3102	3085	3180	3051	3103	31706
5	3518	3349	3283	3323	3367	3260	3252	3227	3228	3216	33023
10	3529	3542	3592	3530	3501	3497	3416	3508	3428	3308	34851
20	3664	3664	3652	3663	3671	3626	3683	3550	3512	3522	36207
30	3927	3927	3927	3952	3997	3858	3757	3849	3757	3762	38713
40	4169	4169	4169	4157	4129	4052	4055	3966	3961	3922	40749
50	4476	4476	4476	4476	4377	4263	4191	4017	4041	4113	42906
$\sum$	35741	35350	35375	35286	35123	34449	34117	33944	33760	33586	346731

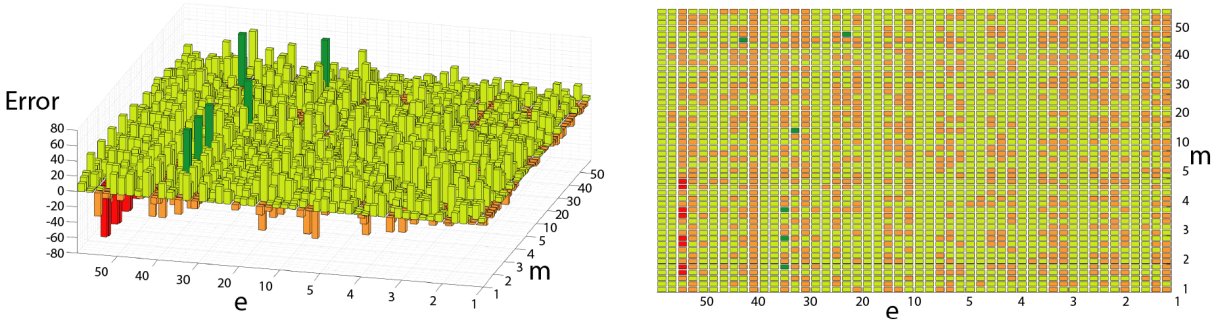

FIGURE 13. Differences of failures with and without LVQ depending on $e, m = 1, 2, 3, 4, 5, 10, 20, 30, 40, 50$.

represent the classification errors while testing combinations of test data sets and test cases according to Table 7.

The colour codes shown in Table 8 distinguish between online and offline data sets to be tested.

Figure 13 shows a fairly homogeneous distribution across all configurations of e and m with a few exceptions. With high values for e and low values for m, the LVQ manipulation sometimes leads to strong deteriorations, e,g. on the online test data set **OnData3** for the last two test cases. In 1783 cases an improvement by LVQ-manipulated LTM is achieved, in 717 cases it leads to a deterioration of the results. Table 9 serves as color code table for Figure 13.

The Tables 10 and 11 show the sum of the classification errors depending on e and m when using the K-Means LTM and LVQ-LTM respectively. Cells with a green background indicate the lowest number of errors per column, cells with the highest number of errors per column are marked in red. The lowest number of errors per row is indicated by a green

font and the highest by a red font. This form of representation clearly shows that the success rate of the classification is higher at low values for e (green/red cells). There is also a tendency that the number of errors decreases with an increase of m (green/red font). Each field shows the amount of misclassifications per 30000 test data vectors, 1200 vectors $\times$ 5 sets $\times$ 5 test cases $=$ 30000 classifications.

V. CONCLUSION

The results of the case study show the robustness of the presented concept for indoor localization via the room impulse response with OL. The classification is highly accurate and can be used for semantic localization in indoor areas with low-cost hardware. Since Brena *et al.* [3] can not find a satisfactory IPS method yet, we recommend further research to make the system applicable in a real environment. The results of the five test cases C1-C5 can be interpreted as follows:

C1) Limitation of the number of prototypes to 13:
The restriction leads to almost the same classification rates compared to the unlimited version, but saves a lot of computing time.

C2) Verification with 19 prototypes per label:
The improvement is not constant on all configurations of e and m. The limitation should therefore be determined considering the application case.

C3) Random order of test data:
With an overall rate of 87.95 % of correct classifications, good results are achieved, so the classification quality of the algorithms does not depend on the insertion of test data in the order of data generation.

C4) Further criterion for the deletion of prototypes:
The change of the deletion strategy does not lead to the desired results, which however underlines the quality of the usage of the *Relative Similarity*.

C5) Only learning with wrong LTM classification:
The change of the deletion strategy does not lead to the desired results too, which gives an indication that not only wrongly classified prototypes, but also prototypes in areas of uncertainty are particularly interesting to learn.

Although the effect of the improved LTM $\mathbf{C}^{\mathcal{L}}_{\mathrm{LVQ}}$ is not clearly visible, the fact that the LVQ-manipulation leads to improved results in 1783 of 2500 cases suggests, that it should be further investigated.

A. DISCUSSION

The OOLA provides good results for the application of a table demo, but raises the question whether the results can also be achieved in larger rooms.

The Tables 10 & 11 clearly show, that low amounts of e (size of the Workbuffer $\mathcal{W}$) lead to better classifications results in our setup. However, it is likely that the e increase in significantly larger test data sets will lead to a long-term improvement in the classification rate, as more candidates that are potentially better suited will be available for selection. The configuration is therefore potentially suitable to configure the algorithm for different use cases. Small values for e could be interesting in cases that are subject to rapid frequent changes. High values for e should be interesting in stable environments, that are subject to minor changes.

Table 10 & 11 also allow the assumption that low values of m (number of learning operations until deletion of STM prototypes with negative *Relative Similarity* relSim^K) can lead to prototypes being deleted "too quickly" and thus not developing their potential, since low values for m often yield poorer classification rates. These effects should also be investigated more intensively in further work.

B. OUTLOOK

Further projects in a 3D environment have already been launched to investigate more complex cases than the ones presented in this work. We are planning an extended case study to generate larger test data sets, a live environment with multiple moving agents to be located and to replace the fingerprinting process with the extraction of an early STM as LTM. The generation of the 7200 raw data files took three working days, so we think about the development of an agent based automated supervised online learning scenario. We continue to think about dynamically modifying the e and m parameters to adjust the learning speed depending on the error rate for different use cases and loading multiple STMs for different situations. In addition, there are some other questions to be explored - especially regarding energy consumption - like the reduction of transmitted characteristics on the IoT-Kit to lower the amount of data, or to use an alternative technology like bluetooth or zigbee instead of WiFi or implementing a threshold monitoring at the analog digital converter to stay in sleep mode till a sound wave reaches the microphone. Also, different room configurations, sizes and distances have to be investigated, not least to determine the required sound pressure corresponding to the range of the process. It might be useful to switch to the infra- and ultrasound spectrum, but this could create new challenges. Finally, the extremely low hardware costs yield great potential for improving the results by investing in better microphones or more sophisticated algorithms.

ACKNOWLEDGMENT

This work was supported in part by Federal Ministry of Education and Research under Grant 01IS17073. Sourcecode and data is available at https://cosy.umwelt-campus.de/software. Parts of this work are based on the master thesis of the first author. Special thanks to Diana Machhamer for the graphical abstract and the visualisations.

REFERENCES

[1] D. Sykes, D. Corapi, J. Magee, J. Kramer, A. Russo, and K. Inoue, "Learning revised models for planning in adaptive systems," in *Proc. 35th Int. Conf. Softw. Eng. (ICSE)*, May 2013, pp. 63–71.

[2] D. L. Silver, Q. Yang, and L. Li, "Lifelong machine learning systems: Beyond learning algorithms," in *Proc. AAAI Spring Symp.*, 2013, pp. 1–7.

[3] R. F. Brena and J. P. García-Vázquez, C. E. Galván-Tejada, D. Muñoz-Rodriguez, C. Vargas-Rosales, and J. Fangmeyer, "Evolution of indoor positioning technologies: A survey," *J. Sensors*, vol. 2017, Mar. 2017, Art. no. 2630413.

[4] M. Dziubany, R. Machhamer, H. Laux, A. Schmeink, K.-U. Gollmer, G. Burger, and G. Dartmann, "Machine learning based indoor localization using a representative k-nearest-neighbor classifier on a low-cost IoT-hardware," in *Proc. 26th Eur. Signal Process. Conf. (EUSIPCO)*, Sep. 2018, pp. 2050–2054.

[5] H. Laux, A. Bytyn, G. Ascheid, A. Schmeink, G. K. Kurt, and G. Dartmann, "Learning-based indoor localization for industrial applications," in *Proc. 15th ACM Int. Conf. Comput. Frontiers*, May 2018, pp. 355–362.

[6] P. M. Hofman, J. G. Van Riswick, and A. J. Van Opstal, "Relearning sound localization with new ears," *Nature neurosci.*, vol. 1, no. 5, p. 417, Sep. 1998.

[7] A. Saxena and A. Y. Ng, "Learning sound location from a single microphone," in *Proc. IEEE Int. Conf. Robot. Automat.*, May 2009, pp. 1737–1742.

[8] L. Fischer, B. Hammer, and H. Wersing, "Combining offline and online classifiers for life-long learning," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2015, pp. 1–8.

[9] G. A. Carpenter and S. Grossberg, "A massively parallel architecture for a self-organizing neural pattern recognition machine," *Comput. Vis. Graph. Image Process.*, vol. 37, no. 1, pp. 54–115, 1987.

[10] C. P. Lim and R. F. Harrison, "Online pattern classification with multiple neural network systems: An experimental study," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 33, no. 2, pp. 235–247, May 2003.

[11] Y. Gu, A. Lo, and I. Niemegeers, "A survey of indoor positioning systems for wireless personal networks," *IEEE Commun. Surveys Tuts.*, vol. 11, no. 1, pp. 13–32, 1st Quart., 2009.

[12] F. Gustafsson, "Particle filter theory and practice with positioning applications," *IEEE Aerosp. Electron. Syst. Mag.*, vol. 25, no. 7, pp. 53–82, Jul. 2010.

[13] E. Vildjiounaite, E.-J. Malm, J. Kaartinen, and P. Alahuhta, "Location estimation indoors by means of small computing power devices, accelerometers, magnetic sensors, and map knowledge," in *Proc. Int. Conf. Pervasive Comput.* Cham, Switzerland: Springer, Aug. 2002, pp. 211–224.

[14] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on Internet of things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1125–1142, Oct. 2017.

[15] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A k-means clustering algorithm," *Appl. Statist.*, vol. 28, no. 1, pp. 100–108, 1979.

[16] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theory*, vol. IT-13, no. 1, pp. 21–27, Jan. 1967.

[17] T. Kohonen, *Self-Organization and Associative Memory*, vol. 8. Springer, 2012. [Online]. Available: https://link.springer.com/book/10.1007/978-3-642-88163-3#about

[18] S. Dörn, *Programmieren für Ingenieure und Naturwissenschaftler: Intelligente Algorithmen und Digitale Technologien*. Berlin, Germany: Springer, 2018. doi: 10.1007/978-3-662-54304-7.

[19] R. J. Kruse, C. Borgelt, F. Klawonn, C. Moewes, G. Ruß, and M. Steinbrecher, *Computational Intelligence: Eine Methodische Einführung in Künstliche Neuronale Netze, Evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze*. Wiesbaden, Germany: Vieweg + Teubner, 2011. doi: 10.1007/978-3-8348-8299-8.

[20] P. Schneider, M. Biehl, and B. Hammer, "Adaptive relevance matrices in learning vector quantization," *Neural Comput.*, vol. 21, no. 12, pp. 3532–3561, 2009.

[21] A. Sato and K. Yamada, "Generalized learning vector quantization," in *Proc. Adv. Neural Inf. Process. Syst.*, 1996, pp. 423–429.

[22] Y. Jin and B. Sendhoff, "Alleviating catastrophic forgetting via multi-objective learning," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, Jul. 2006, pp. 3335–3342.

[23] L. Fischer, B. Hammer, and H. Wersing, "Efficient rejection strategies for prototype-based classification," *Neurocomputing*, vol. 169, pp. 334–342, Dec. 2015.

[24] L. Fischer, B. Hammer, and H. Wersing, "Certainty-based prototype insertion/deletion for classification with metric adaptation," in *Proc. ESANN*, 2015, pp. 1–6.

[25] M. Grbovic and S. Vucetic, "Learning vector quantization with adaptive prototype addition and removal," in *Proc. Int. Joint Conf. Neural Netw.*, Jun. 2009, pp. 994–1001.

[26] S. Kirstein, H. Wersing, and E. Körner, "Rapid online learning of objects in a biologically motivated recognition architecture," in *Proc. Joint Pattern Recognit. Symp.* Cham, Switzerland: Springer, 2005, pp. 301–308.

RÜDIGER MACHHAMER received the master's degree in computer science from the Trier University of Applied Sciences, Environmental Campus Birkenfeld, Germany. He is currently pursuing the Ph.D. degree with the Research Group Distributed Systems, Institute for Software Systems (ISS), where he is involving in research projects COSY and IoT-Pilot, ISS. His research interests include machine learning, online learning, and the Internet of Things.

MATTHIAS DZIUBANY received the B.S. degree in mathematics and the M.S. degree in economic mathematics from Technical University Kaiserslautern. He is currently pursuing the Ph.D. degree in electrical engineering and information technology with RWTH Aachen University. He is also with the Research Group Distributed Systems, Institute for Software Systems (ISS), Environmental Campus Birkenfeld. His research interests include machine learning, optimization, and game theory.

LEVIN CZENKUSCH received the B.Sc. degree in computer sciences from the Trier University of Applied Science, Environmental Campus Birkenfeld, in 2018. He is currently pursuing the M.Sc. degree with the Research Group Distributed Systems, Institute for Software Systems (ISS), Environmental Campus Birkenfeld.

HENDRIK LAUX received the M.Sc. degree in electrical engineering from RWTH Aachen University, in 2019. He is currently pursuing the Ph.D. degree with the ISEK Research Area (RWTH Aachen) on machine learning and Ai for intensive care medicine.

ANKE SCHMEINK received the Diploma degree in mathematics with a minor in medicine and the Ph.D. degree in electrical engineering and information technology from RWTH Aachen University, Germany, in 2002 and 2006, respectively. She was a Research Scientist with Philips Research before joining RWTH Aachen University, in 2008, where she has been currently an Associate Professor, since 2012. She spent several research visits with the University of Melbourne and the University of York. Her research interests include information theory, systematic design of communication systems, and bioinspired signal processing. She is also a member of the Young Academy, North Rhine-Westphalia Academy of Science.

KLAUS-UWE GOLLMER received the Diploma degree in biomedical engineering from the University of Applied Sciences, Hamburg, in 1987, the Diploma degree in electrical engineering from Technical University Hamburg, Harburg, in 1991, and the Ph.D. degree from Leibniz University Hannover, in 1996. Since 1999, he has been a Professor of modelling and simulation with the Trier University of Applied Science. His research interests include the Internet of Things and machine learning. He is also a member of the IoT Expert Group of the German National Digital Summit.

STEFAN NAUMANN received the degree in computer science from the Universities of Kaiserslautern, Saarbrücken, Germany, and the Ph.D. (Dr.rer.nat.) degree in natural sciences from the University of Hamburg, in 2006. He is currently a Full Professor of fundamentals in computer science, mathematics, and environmental and sustainability informatics with the Trier University of Applied Sciences, Environmental Campus Birkenfeld, Germany and a Directorate Member of the Institute for Software Systems (ISS). He has authored over 70 peer-reviewed articles in this field. His research interests include sustainable development in conjunction with computer science and the environmental impacts of information technology, especially of software.

GUIDO DARTMANN received the Diploma degree and the Ph.D. degree from RWTH Aachen University, in 2007 and 2013, respectively. Since 2016, he has been a Professor of distributed systems with the Trier University of Applied Science. His major research interests include distributed systems, hardware software co-design and machine learning. He is also a member of the IoT Expert Group of the German National Digital Summit and a Founding Member of the IEEE Special Interest Group on Big Data Intelligent Networking.