International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 6 Issue: 3 223 - 229

Context Aware Sensor Collaboration for Intelligent Wireless Communications
an AI Approach to Moving Sensor Managemet

John Yoon
Math/CS Department
Mercy College
Dobbs Ferry, New York, USA
jyoon@mercy.edu

***Abstract*—**Collaborative sensor data service is an emerging technology and it is beneficial to various applications including robotics, medicals, industry and military. Sensor collaborations improve technical difficultiesonthe verification and validation of sensor data or reduction of wireless sensor data transmission. However, typical approaches to sensor collaborations are less satisfactory. It is in part because the sensor calibrations are pre-fixed and therefore they are less adaptive to dynamic changes in environment. It is also because sensors are calibrated one time before deployment, and their collaborations do not take into consideration the dynamic movement of sensors. Their calibrations are not satisfactorily adaptive to environmental changes, or their collaborations are less efficient to cope with abrupt presence/absence of sensors. This paper proposes a two-tier deep learning technique to enable sensor devicesto be adaptive and moving sensors to be collaborative. The contribution of this paper is an intelligent identification of environment changes and intelligent rearrangement of wireless sensor network.

Keywords*-Artificial Neural Network, Genetic Algorithm, Collaborative Wireless Sensor Network, Sensor Data Calibration*

*****

I. INTRODUCTION

A wireless sensor network is a wireless network consisting of spatially distributed autonomous devices using sensors to sense environmental conditions. Sensors are widely used to collect information about their environment, and the sensor data sets are further analyzed in a high computing device. Sensor data is the output of a sensor device, in some type, converted from the type of input data sensed about a target environment. The type of a sensor device is converted to a data form that can be read by a microcontroller unit (MCU). A MCU usually has software packages running to acquire sensor data sets from various sensor types. The first level of data analytics may be done at a MCU, however the main analytics of sensor data can be done at a sensor analytics server (SAS) as illustrated in Figure 1.

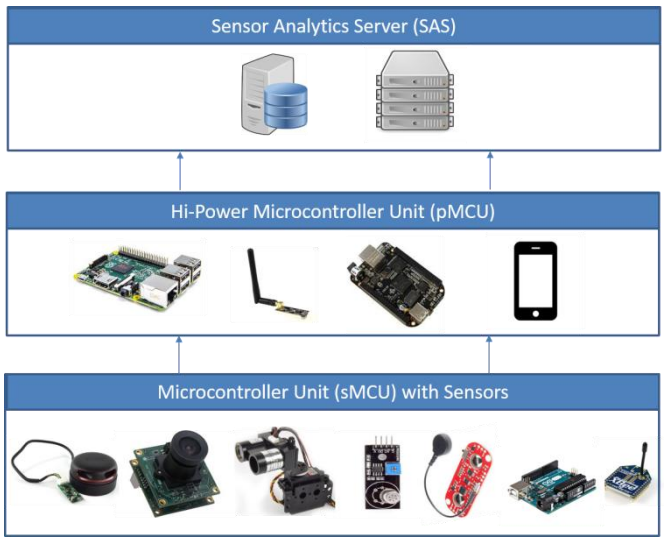


Figure 1. Wireless Sensor Networks

As shown in Figure 1, Sensor data processed at a MCU are those acquired by the sensors that are installed at that MCU. However, sensor data processed at a SAS are those acquired by various sensors, each of which is installed at a different MCU. For example, MCU1 has an infrared (IR) camera and an audio detector, and MCU2 has an IR camera and a motion detector. Each sensor collects sensor data, which are then transmitted to a SAS. At a SAS, there are data values from two camera (one IR and another non-IR), audio detector and motion detector, which are collaborated. Figure 1 illustrates three tiers of sensor data services. As an example on the lower tier, there are MCUs on which wireless sensors are deployed and sense environment data. On the middle tier, there are MCUs which are empowered with high speed CPUs and bigger memory spaces. For example, Arduino UNOs with distance sensors are deployed on the lower tier, while Raspberry Pi or Beaglebone board with robust network devices are installed on the middle tier.

Of course, Bluetooth communications are also possible. On the top tier, computing power units are available such as smartphones and servers.

Sensors should be calibrated, and sensor data are collected by a calibrated sensor. The sensor calibration determines the sensitivity and granularity of data to be collected. For example, cameras collect image data when lenses are correctly focused and distances are properly tuned. Sound detectors can collect acoustic data accurately when the distance to a target object is accurately set. Motion detection sensors can collect object movement states accurately if the sensitivity of a target object and the triggering time are properly adjusted.

Sensor data is one of the sources that can raise the size of big data, and it improves sensor data services exponentially. Sensor device calibration and sensitivity adjustment determines the

quality of sensor data services. Sensors or MCUs with sensors are remotely deployed. When deployed, sensors may be properly calibrated and adjusted to target objects or target environments.

A. *Problem Description*

Sensor data may not be accurate due to pre-set calibration which does not reflect the current environment. Although calibration and adjustment is set perfectly once, it may not be effective continually. Even if there is a way of calibrating sensors remotely, it is an expensive process or its calibration cannot be made timely. The problem that this paper describes includes the following:

- Remotely deployed sensors are uneasy to calibrate especially as sensor collaborations are needed.
- Once deployed, sensors data are uneasy to adapt to remote situation and context changes.

B. *Motivating Example*

Preset calibration for sensors does not measure what the real changes in environments occur. As shown in Figure 2(a), sensors installed in a hospital ward may be able to identify patient data.

However, the very same sensor calibrated in the same way in hospital wards should not work accurately if it is deployed at Time Square in NYC (See Figure 2(b)). It is simply because the environmental noises between the two locations is not similar if not quite different.

Uniform calibration for sensors does not identify the changes of environment. For example in Figure 2(c), there are two situations, each developed abruptly from those in Figure 2(a) and 2(b). Hospital wards and Time Square may have higher noises due to attacks. The identification of any environmental data may not be made if the initial calibration last unchanged.

(a) Hospital Environment

(b) Time Square Environment

(c) Attacks on (a) and (b)

Figure 2. Motivating Examples

C. *Approach Sketch*

To resolve or overcome the problems and difficulties stated above, this paper proposes a novel approach as depicted in Figure 2. In the figure, the three tiers architecture of smart computing on the left, and our proposed approaches to make sensor data services intelligent on the right. Those two approaches are briefly described:

- An artificial neural network (aNN) technique approach to sensor data calibration: Sensors of the same sensor type can be trained to adjust a target environment. Adjustment of sensor calibration is trained by an artificial neutral network at each MCU. Once each sensor is deployed at a site (MCU) and trained well for collaboration with other sensors in the same MCU, it starts to acquire sensor data.
- A genetic algorithm (GA) approach to sensor data interpretation: Multiple series of sensor data, each series transmitted from a MCU, are analyzed at a SAS. Since a few of sensors are active or inactive occasionally, the software packages installed on MCUs or SASs should be adaptive to such missing or present sensors. Sensor data series are crossover, while a mutation of sensor data may occur in a sensor data series. This process can be managed and predicted by using a genetic algorithm.

The contribution of this paper includes

- Sensors can be calibrated at real time. A way of sensor calibration is trained to be adaptive to target environments.
- Sensor collaboration becomes intelligent to identify the data states of unforecastable sensors. It is likely that some sensors are inactive to disappear or all of sudden active to appear.

D. *Paper Organization*

The remainder of this paper consists of the following sections. Section 2 introduces a few sensor technologies with wireless sensor networks. Techniques of artificial neural networks and genetic algorithms are also reviewed. Section 3 describes the model of sensors and sensor data over wireless sensor networks. MCU and SAS are defined based on sensor data in this section. Section 4 describesartificial neural etwork to train and practice sensor calibrations. Sections 5 describes a genetic algorithmic approach to the collaboration of moving sensors. Section 6 shows an evaluation. Section 7 concludes this paper

II. Background and Related Work

A. Wireless Sensor Network

Sensor data can be rapidly and continuously fed from various sensors: data monitoring scientific or environmental changes, data measuring health care and patients, data about social media sharing, etc. Sensor data is generated infinitely real-time and needs to collaborate analytically to make a better decision in cloud server [1], Fog Computing [2], Mobile Cloud Computing (MCC) or Mobile-Edge Computing (MEC) [3].

As sensor data are used in Fog Mobile-Edge Computing (FMEC) [4], the quality of sensor data need to be managed. Sensor devices may participate or disappear for many reasons such as energy discharge. Malicious sensor devices may participate or replace trusted sensors. Not only sensor devices, but data acquired by sensors may be another issue to verify its quality or validity.

There are two types of sensor data acquisition: passive and active sensor data acquisition. Sensors acquire environmental or monitorial data actively or passively. Sensor data can be actively acquired by taking into consideration the differences between transmitting and returning signals. On the other hands, sensor data can be passively acquired as signal is emitted from environment. For example, motion detection sensors or proximity sensors emit signals to an object and receive the returning signals from the object. This type of sensors is called an active sensor. The light detection sensors or hall magnetic sensors, which generate no signals, take light or magnetic energy, which will be then converted into an electrical energy. This type of sensors is called a passive sensor.

While sensor data transmitted from various sensors are analyzed collaboratively, the accuracy or the reliability of analytic outcome is determined by the quality of the sensor data.

B. Artificial Neural Network

Neural network techniques are used to classify sensor data in several areas such as sound [5] and smell [6]. The C4.5 decision tree classifier is employed based on the features of the sound wave spectrogram [7]. Parameterized features extracted from sound wave data, which are acquired in wireless sensor networks, are classified using artificial neural network.

One of the weakness of wireless sensor network is the limited energy of the sensor node. Reduction of sensor data transmission and its prediction is also studied using artificial neural network [8].

Sensor data may be missing due to several reasons such as disconnection to a MCU or even further to a SAS. Such missing sensor data can be obtained by neural network [9]. Training the neural network on sensor data with missing values artificially filled in has been introduced.

One of the advanced technologies in artificial neural network is a deep artificial neural network or context-dependent machine learning [10]. The approach that this paper proposes is also to leverage target context to calibrate sensors and sensor data as shown in Figure 3.

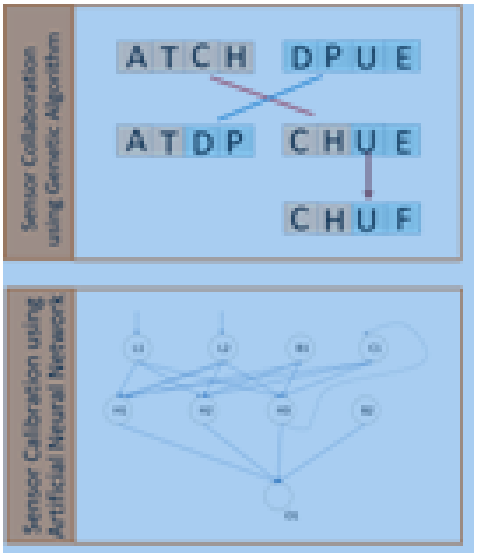


Figure 3. Two-tier deep-learning sensor networks

C. Genetic Algorithm

Genetic algorithm (GA) is an algorithm, introduced by John Holland at University of Michigan in 1970s, that mimics some of the processes observed in biological evolution and provides the steps to compute the biological evolutionary processes [11]. GA simulates the survival of the fittest among objects (or things or *solutions*) over the course of generation to generation evolutions.

GA applies two major operations: *mutation* [12] and *crossover* [13]. These GA operators are applied to population of objects (or solutions) or their offspring. Similar to biological evolution process, the mutation or crossover operators take place randomly. When a mutation occurs, it occurs only a small spot of an object (or solutions).

Similar to the evolution in the chromosome of DNA, each MCU defines sensor data lists (or solutions) to represent the problems to solve. Those solutions are then transmitted from MCUs, from which a SAS selects few solutions. Only those sensor data lists (or solutions) most successful in each sensor data service will produce more offspring as shown in Figure3.

III. Sensor Data Model

A. Sensors and Microcontroller Units (MCUs)

A MCU has sensors, which are denoted as letters in the sensor set {A,B,…,Z}. For simplicity, this paper denotes each letter as a sensor type, and it denotes a letter with subscript as a sensor data. So,

$$MCU = \{ x \mid x \text{ is in sensor data} \} \quad (1)$$

For example, at a particular time, MCU1 may transmit a sensor data list, [*a*1,*c*1,*t*1,*h*1]. This model is illustrated in the lower level sMCU's of Figure 1.

B. Sensor Data at Sensor Analytics Server (SAS)

Sensor data at a SAS is a set of sensor data lists received from a MCU. So,

$$SAS = \{ x \mid x \text{ is a MCU}\} \quad (2)$$

In Figure 1, the top levels, SAS or pMCU collects sensor data, and so it can be a set of sMCU's.

For example, a SAS may receive a sensor data list [*a*1,*c*1,*t*1,*h*1] from MCU1, and [*a*2,*d*2,*e*2,*g*2] from MCU2 now. At the next time, the sequence would be [*a*1,*c*1,*t*1,*h*1], [*a*2,*d*2,*e*2,*k*2] and [*c*3,*d*3,*e*3,*l*3,*p*3] from MCU1, MCU2 and MCU3, respectively.

.ANN Approach to Sensor Calibration

As sensors are deployed at a target environment, tuning sensors and interpreting sensor data is a hard problem. Manual and remote calibration of sensors or pre-set calibration has drawbacks if not inefficient. To improve sensor calibration that can be adaptive to a given target environment, this paper proposes artificial neural network.

In the training phase of ANN as illustrated in Figure 4, for given sensors, a sensor data [*a*1,*c*1,*t*1,*h*1] can be first considered, which is the input to the ANN together with the information, *T*1, about a given target environment. There are intermediate nodes, H_i's, and output node *O*1. In addition to that, if needed, bias nodes, B_i's, and feedback nodes, F_i's can be involved as shown in the figure. These nodes appear on the input layer, hidden intermediate layer, or the output layer. The number of hidden intermediate layers in ANN is usually very high (i.e., multi-million times), and it will be optimized or stops when no more significant update is made.

We used this ANN to find the relation between the given calibration parameter of sensors and the target environment. For example, a piezoelectric acoustic wave sensor *a*1 has parameters about delay lines, resonators, wave frequencies, etc [14]. We take a wave frequency 1.2 GHz as input for *a*1. A Raspberry camera module *c*1 has parameters such as photo resolution, video resolution, time elapse, focal length, focal ratio, etc (in part, refer to [15]). We take a focal length 9m as input for *c*1. A motion detection sensor *t*1 has parameters, detection range, radio frequency, light density for measuring, etc [16]. We take a detection range 30m as input for *t*1. An inductive proximity sensor *h*1 has parameters, operating distance, detection angle, etc [17]. We take an operating distance, 0.03m as input for *h*1. Note that *h*1 is installed near *a*1, so that noise sound generated from a bug, which is near *a*1, can be detected by *h*1 and therefore the noise can be filtered out.

In addition to the input for the sensors, a target environment will be of an object such as a victim of accidents, a patient in hospital rooms, or an animal in wild areas. The quality of sensor data sets depends on the characteristics of a target environment. The parameters of target environments include the minimal bound rectangle of an object, the speed of an object, the thermal image of an object, etc [18]. Assume the target environment data set T = {*ca, md, fg, bm*}, which respectively denote car accident, murder incident, gun firing, bomb explosion.

Now, we can consider a few of the input to our ANN as shown in Figure 4. The input will be a sensor data set plus the target environment data.

Sensor Data Lists at Input Layer:
{[1.2, 9, 30, 0.03, *ca*], [0.5, 25, 31, 0.5, *md*], [5.7, 2, 250, 0.03, *gf*], …, [1.3, 9, 28, 0.029, *bm*]}

The train to this input will be
[1,0,0,…,1],
where 0 and 1 respectively denote negative and positive example. Of course the bias and feedback can be considered in this ANN.

Hidden layers: Multiple hundred millions of iteration can take place. There will be multiple activation functions that can be used from an upper hidden layer to a lower hidden layer. The output of this neural network will then the classification of the input set.

The classification as the output of this neural network will be 1) the calibrated sensor data list and 2) the one disqualified sensor data list.

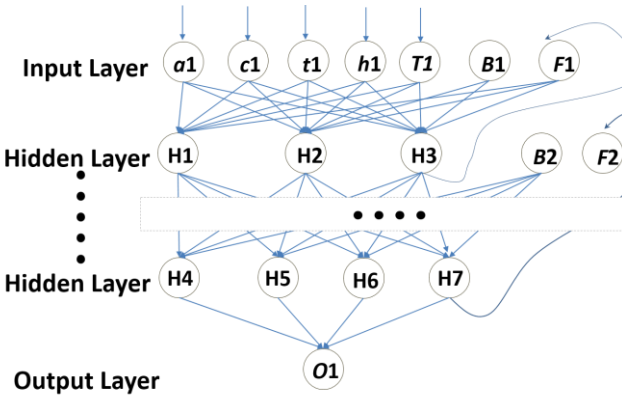


Figure 4. ANN approach to sensor calibration

Output Layer: The following is a subset of the output.

- Calibrated Sensor Data List: {[1.2, 9, 30, 0.03, *ca*], [0.5, 12, 31, 0.5, *md*], [3.1, 2, 50, 0.03], …, [1.3, 9, 28, 0.029]}. What each of the sensor data lists means that the sensor collaboration with those parameters is acceptable. For example, the first sensor data list means that the acoustic data in 1.2GHz for camera image with 9m focal point, 30 meter detected motion, 30cm proximity will work for car incidents. This is a calibrated sensor data list that is learned from ANN.
- Disqualified Sensor Data List: {[0.5, 25, 31, 0.5], [5.7, 2, 250, 0.03, *gf*], …, [1.3, 91, 28, 0.029, *bm*]}. This sensor data list contains sensor data lists that a SAS may ignore for further analysis or report errors.

After the training phase of the neural network, the testing or practicing phase comes. At testing or practicing phases, those calibrated sensor data lists will be directly used to transmit the data to a SAS.

IV. GA APPROACH TO SENSOR COLLABORATION

This section describes two major operators used in genetic algorithms: crossover and mutation. It is likely that the fitness or goodness function is determined by taking the application specific issues into consideration. Each application has different issues to make sure its goodness or fitness.

The sensor model defined in Section III, a MCU is a set of sensor data (refer to Equation (1)). As an example, in order to define the fitness of wireless sensor network, consider a code segment implemented in an Arduino, which acquires sensor data from a dust sensor, a temperature sensor and a distance sensor.

Sample code segment in Arduino

```
(1)  #include <SPI.h>
(2)  #include "RF24.h"
(3)  struct package {
(4)    float dustDensity;
(5)    float temperature;
(6)  int distance;
(7)  };
(8)  typedef struct package sensorData;
(9)  sensorData data;
(10) void setup() {
       // omitted
(11) }
(12) void loop() {
(13)    data.dustDensity = analogRead(pin5)
(14)    data.temperature = analogRead(pin7)
(15)    data.distance = analogRead(pin9)
(16)  Serial.println(data);
(17) }
```

Note that lines (3)-(7) is a struct data type in C. As stated in the struct data type, three sensors are included in a MCU. The fitness function in this case can be defined as follows:

$$\text{Fitness}(S) = \sum_{1}^{m} \|M_i\| \qquad (3)$$

where for $S = \{M_1, M_2, \ldots, M_m\}$ and $M_i = \{s_1, s_2, \ldots, s_n\}$, which are respectively from Equation (2) and (1), $\|M\|$ is the normalized values, i.e., $\|M_i\| = \prod_1^n s_j$. If a sensor s_j is calibrated in the range $[l..h]$ and the sensor data acquired is k, then $s_j = \frac{k-l+1}{h-l+1}$.

Sensor data acquired by MCUs can be transmitted to upper layers in Figure 1 and can arrive at a SAS. The following code segment of Python is to relay sensor data to upper layers.

Sample code segment to transmit sensor data at real time

```
(1) import sys, re, wirelessSN
(2) from datetime import datetime
(3) textIn = sys.argv[1]
(4) wirelessSN.begin()
(5) wirelessSN.openWritingPipe()
(6) for line in sys.stdin:
(7)   if re.search(textIn, line):
(8) wirelessSN.write(line)
(9) wirelessSN.write(datetime.now().strftime('%Y-
    %m-%d') + "\t"+line+"\n")
```

Line (3) is a keyword of our interest if any that can be to search the sensor data (in line (5)). Then, the line of sensor data is shipped out in lines (8) and (9).

A. Crossover in Sensor Sequences

Recall the model of sensors described in Section 3, a sensor data sequence is a sequence of characters, which is a string. The crossover operator can apply in a straightforward way as follows:

One half the characters of a sensor data sequence are crossed over with the other half of another sensor data sequences. The outcomes should be confirmed if they are also available in the list, *List*(x). If the crossover operator takes place on SAS1, the outcome should be in *List*(SAS1).

For example, assume that List(SAS1) = {$a1$,$c1$,$d1$,$e1$,$g1$, $l1$,$m1$,$n1$,$p1$,$s1$}. Consider a sensor data sequence, {[$a1$,$t1$,$c1$,$h1$], [$d1$,$d1$,$u1$,$e1$]} that is received at SAS1. As shown in Figure 3, the first half is replaced by the second half of the other sensor data.

B. Mutation of Sensor Sequences

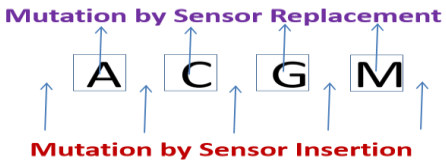


Figure 5. Mutations of sensor

Mutation probability is a measure of the likelihood that random characters of sensor data in sensor data sequences will be modified into the data of some other sensors. Note that the characters modified should be also in the list of the sensors deployed, *List*().

Mutation, which is a random change, takes place less likely than the crossover. A sensor data character can be mutated if a change occurs on

- one or more characters of the symbol, and/or
- one or more spaces in between.

The former is called 1) *replacement-based* sensor mutation, and the latter is called 2) *insertion-based* sensor mutation as illustrated in Figure 5.

For example, in Figure 5, the sensor data [$a1$,$c1$,$g1$,$m1$] is mutated to [$a1$,$s1$,$g1$,$m1$] by replacing character C by S, meaning that sensor C is replaced by the new sensor S. If a sensor N is deployed or re-activated in a MCU, the outcome of an insertion-based sensor mutation will be [$a1$,$c1$,$g1$,$m1$,$n1$].

An implementation of the proposed mutation techniques is shown below:

Python Coding list for Mutation

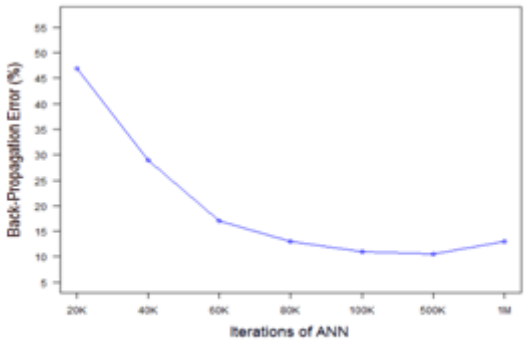
```
def mutate(aParent):
  symNo = len(aParent)
  indx2mut = random.randint(0, symNo) - 1
  sym2mut = aParent[indx2mut].strip()
  sizeSym = len(sym2mut)*2
    mutPosition = random.randint(0, sizeSym)

if mutPosition % 2 == 1: // replacement based
    index = int(mutPosition / 2)
    modifi = random.choice(string.ascii_letters +
string.digits)
    while sym2mut[index] is modifi:
      modifi = random.choice(string.ascii_letters +
string.digits)
    symMutated = sym2mut[:index] + modifi +
sym2mut[index+1:]

else:  // insertion based
    index = int(mutPosition / 2)
    symMutated = sym2mut[:index] +
random.choice(string.ascii_letters + string.digits) +
sym2mut[index:]

return symMutated
```

The above program module generates a mutation based on a random number generated. A choice between sensor replacement and sensor insertion is made by the random number. The position selection or the character selection for replacement or insertion are also made by the random number generation.

V. EXPERIMENTAL RESULT

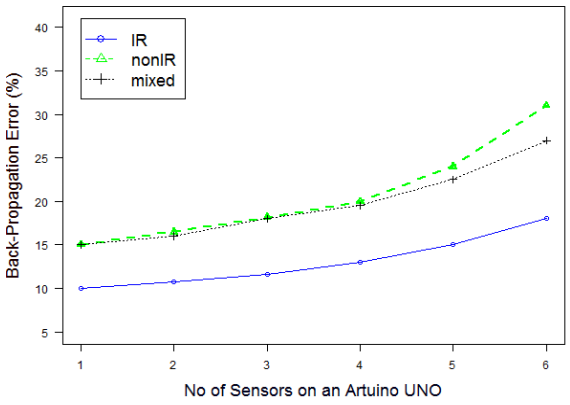
As illustrated in Figure 1 and Figure 3, a small collection of sensors is controlled by a MCU and a collection of sensor data received from MCUs is analyzed in a SAS. Over the course of three tiers, artificial neural networks and genetic algorithms are employed to improve the quality of sensor data. It is in part because sensors do not consistently participate in acquiring and transmitting sensor data. MCUs and SASs need to be more intelligent to cope with such a dynamic situation.

For the training phase of ANN, sensor data sets are collected from the Dobbs Ferry campus lab, which is a quiet residential suburban area, and the Manhattan campus lab, on Broadway and 35th Street, which is one of the busiest area. At each lab experience, 100 sensors and 50 MCUs are used for various combinations and testing. We examined both IR and non-IR sensors, and Arduino UNOs only for MCU. In our experiments, we evaluate the error [19] that appears on feedback propagation, denoted as the node F, in Figure 3.



(a) wrt Iteration of ANN

In the first set of experiments, the iterative execution of ANN loop is experimented by increasing the number of iterations. As increasing the layers of hidden nodes, the test of the errors caused from the fact that SASs do not recognize whether any sensor devices become presence. Figure 6(a) shows that there is an optimal iteration where the ANN-based



(b) wrt Sensor Type

Figure 6. Error Evaluations

classification splits the class of environment-well-adapted sensor data sequences and the class of invalid or less-likely useful sensor data sequences. Each application for different environment determines different optimized iteration, which indicates the lowest error in the figure. It is unlikely that the more iteration, the better training.

In the second sets of experiments, the back-propagation errors in ANN are compared based on varying the ratio of sensors on MCU. Figure 6(b) shows three settings over Arduino UNOs (MCUs): 1) IR sensors only on MCUs, 2) non-IR on MCUs, and 3) mixed sensors. One thousand iterations of ANN are executed. IR sensors only on MCUs are trained the best with the lowest error.

In the third sets of experiments, two different environments are considered: one at Dobbs Ferry campus lab with the low noise below 40dB, and another at Manhattan campus lab with the high noise over 80dB. The experimental scenario the third evaluation is to sense accidents occurring on streets. Consider sensors: surveillance cameras, sound detection sensors, proximity sensors, and motion detection sensors. Various different combinations among them are installed on Arduino UNOs. Each MCU collects sensor data and transmits them to a Raspberry Pi as SAS.

[App1] If no intelligent software is running on MCUs, all sensor data that are acquired by sensors are transmitted.

[App2] If a sound detection sensor detects the noise from an accident, the surveillance camera images are transmitted.

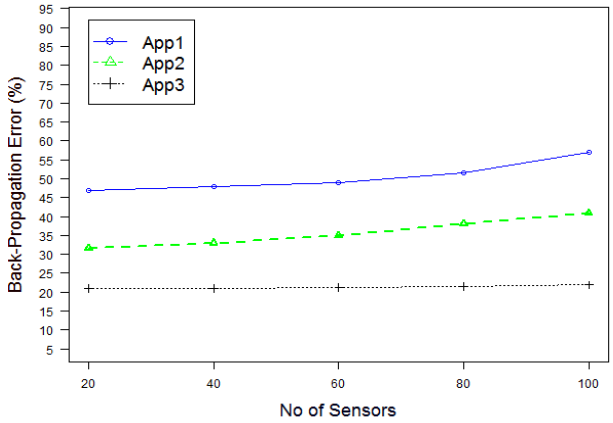
[App3] If a software piece runs to determine the truth of noises intelligently and adaptively, proper sensor data sets only are transmitted.

[App4] Although App3 is enabled, there are sensors that can disappear or back in active abruptly. At a SAS, GA algorithms are employed to cope with mutations of sensor data.

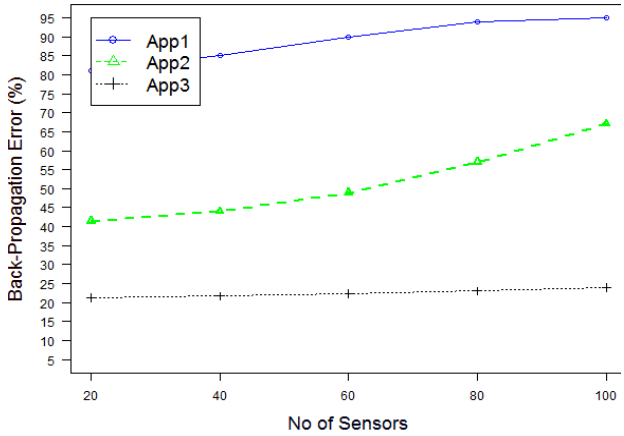
For the above four approaches, back-propagation errors are measured as sensors increase. Note that GA algorithms are employed only on App4. Since the back-propagation errors are measured, Figure 7(a) and (b) show the error comparison on

 ISSN: 2321-8169
223 - 229

App1 - App3 only. App1 produces more errors than others, and



(a) Sensors in Quiet Environment (< 40dB)



(b) Sensors in Noisy Environment (> 80dB)

Figure 7. Error Evaluations

it is especially worst when the base noise is high like 60dB or higher. App3 that employs ANN becomes very tolerant to the changes in target environment.

VI. CONCLUSION

This paper described a deep learning approach to 1) calibrate sensors and 2) enable sensors to be collaborated. Sensor calibration is not prefixed but is adaptive to environment changes. Sensor collaboration and their arrangement is not predefined but dynamically defined based on sensor availability. When it comes to moving sensors to be deployed, moving sensors are not necessarily belonging to a specific MCU. A deep learning technique identifies appearance and disappearance of sensors. Sensors and sensor data are formalized for an artificial neural network and a genetic algorithm. An artificial neural network is used to calibrate sensor data on MCUs. A genetic algorithm is used to improve sensor data sequences at SASs to cope with abrupt changes of sensor data for some unknown reasons.

The two-tiered deep learning, our ANN approach followed by GA algorithm execution, improves the calibration of sensors and that approach also maintains the sensor data more adaptive to environments with both high and low noises.

REFERENCES

[1] S. Bose, A. Gupta, S. Adhikary, N. Mukherjee, “Towards a sensor-cloud infrastructure with sensor virtualization,” Proc. Of the 2nd Workshop on Mobile Sensing, Computing and Communication, pp. 25-30, 2015.

[2] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, “Fog Computing and Its Role in the Internet of Things,” ACM SIGCOMM 2012, pp. 13- 15, August, 2012.

[3] P. Lopez, A. Montresor, D. Epema, A. Datta, , A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, pp.68–73, 1892.

[4] S. Yi, C. Li, Q. Li, “A survey of Fog Computing: concepts, applications and issues,” in Proc. Of the 2015 Workshop on Mobile Big Data, pp. 37–42, 2015.

[5] J. Colonna, T. Peet, C. Ferreira, A. Jorge, E. Gomes, J. Gama, “Autoatic classification of anuran sounds using convolutional neural networks,” Proc. Of the 9th Int’l C* Conf. on Computer Science & Software Engineering, 2016.

[6] S. Omatu, M. Yano, “Smell classification using weakly responding data,” Proc. Of the 29th Annual ACM Symposium on Applied Computing, 2010.

[7] W. Hu, N. Bulusu, C. Chou, S. Jha, A. Taylor, V. Tran, “Design and evaluation of the hybrid sensor networks for cane toad monitoring,” ACM Transactions on Sensor Networks, 5(1): 4, 2009.

[8] S. Zhang, Q. Zhang, S. Xiao, T. Zhu, Y. Gu, Y. Lin, “Cooperative data reduction in wireless sensor network,” ACM Transactions on Embedded Computing Systems, 14(4), 2015.

[9] L. Wong, H. Chen, S. Lin, D. Chen, “Imputing missing values in sensor networks using sparse data representations,” Proc of the 17th ACM Int’l Conf on Modeling, Analysis and Smulation of Wireless and Mobile Systems, pp. 227-230, 2014.

[10] P. Bell, P. Swietojanski, S. Renals, “Multitask learning of context-dependent targets in deep neural network acoustic models,” IEEE/ACM Transactions on Audio, Speech and Language Processing, 25(2), 2017.

[11] D. Goldberg, Genetic Algorithms in search, optimization, and machine learning, Addison-Wesley, 1989.

[12] S. Nareddy, E. Westover, K. Hillesland, W. Kim, “Genome dynamics in coevolved genomes: database management system for tracing mutations,” Proc. of the 5th ACM Conf. on Bioinformatics, Computational Biology and Health Informatics, 2014, pp. 633-634.

[13] J. Bogard, “A probabilistic functional crossover operator for genetic programming,” Proc. of the 12th Annual Conf. on Genetic and Evolutionary Computation, 2010, pp. 925-931.

[14] How SAW Sensors Operate?, from http://www.senseor.com/saw-technology/saw-sensors-operation, on April 3, 2017.

[15] Raspberry Camera Module v2, from https://www.sparkfun.com/products/14028, on March 20, 2017.

[16] FIBARO Motion Sensor, from http://manuals.fibaro.com/content/manuals/en/FGMS-001/FGMS-001-EN-T-v2.0.pdf, on February 25, 2017.

[17] Operating Principles for Inductive Proximity Sensors, from http://www.fargocontrols.com/sensors/inductive_op.html, on March 17, 2017.

[18] Total Body Thermography, from http://total-body-thermography.com/, on March 31, 2017.

[19] D. Rumelhart, G. Hinton, R. Williams, “Learning representations by back-prpagating errors,” Nature, 323: 533-536, 1986.