Opportunistic Sensing in Wireless Sensor Networks

Hans Scholten and Pascal Bakker
Pervasive Systems
University of Twente
Enschede, the Netherlands
hans.scholten@utwente.nl, aboe@aboe.nl

***Abstract*—Opportunistic sensing systems consist of changing constellations of wireless sensor nodes that, for a limited amount of time, work together to achieve a common goal. Such constellations are self-organizing and come into being spontaneously. This paper presents an opportunistic sensing system to select a subset of sensor nodes out of a larger set based on a common context. We show that it is possible to use a wireless sensor network to make a distinction between carriages from different trains. The common context in this case is acceleration, which is used to select a subset of carriages that belong to the same train. Simulations based on a realistic set of sensor data establish that the method is valid, but that the algorithm is too complex for implementation. Downscaling reduces the number of processor execution cycles as well as memory usage, and makes the algorithm suitable for implementation on a wireless sensor node with acceptable loss of precision. Actual implementation on wireless sensor nodes confirms the results obtained with the simulations.**

***Keywords*-opportunistic sensing, wireless sensor network, context awareness, activity recognition**

I. Introduction

”Opportunistic sensing is seen as a way to gather information about the physical world in the absence of a stable and permanent networking infrastructure.” (Opportunity Workshop at Ubicomp 2010, Copenhagen, Denmark). The absence of a stable and permanent networking infrastructure dictates that collected information is either processed and acted upon inside the network by opportunistic collections or clusters of nodes [1], or the information is preproceesed and stored inside the network untill there is an opportunity to forward it outside the network, as is the case in delay tolerant networks [2], [3], [4], [5].

Opportunistic sensing or networking is often associated with human-centric ubiquitous systems, such as in crowd sourcing and participatory sensing applications [6], [7], [8] or are focusing on human activity recognition [9], [10], [11].

In this paper we show how context awareness based on a common pattern of movement is used to select only those carriages from a much larger population that belong to the same train. Movement as a discriminating factor for context awareness has been described earlier [12], [13], but not for trains. The problem was first introduced in [14] describing a communication protocol for wireless networks in linear structures such as trains. The network is part of a railway safety system to monitor the initial composition of a train and to detect any change in composition once the initial composition is established. In Europe many different safety systems exist and trains crossing borders must be equipped with all applicable safety systems. These systems are infrastructure based and integrated in the tracks, whereas the proposed system is entirely train based. Composing a train not only takes place on switchyards, but also on the way when carriages are added to the train or decoupled from it. Although a simple beacon based detection system seems appropriate, it would not suffice to check the train’s composition. Because the ID of the beacons in sight are not known a priori, nor the mapping of IDs to carriages, nor carriages to trains, there is no way to discriminate carriages in different trains in close proximity. Additional measures must be present to select a train’s beacons from all beacons that are in communication range. In this paper, motion is used as the discriminating feature between trains.

In the remainder of this paper we will discuss the collection and analysis of the data sets to be used in the simulations, the simulation itself and the implementation on wireless sensor nodes respectively, followed by a discussion of the results.

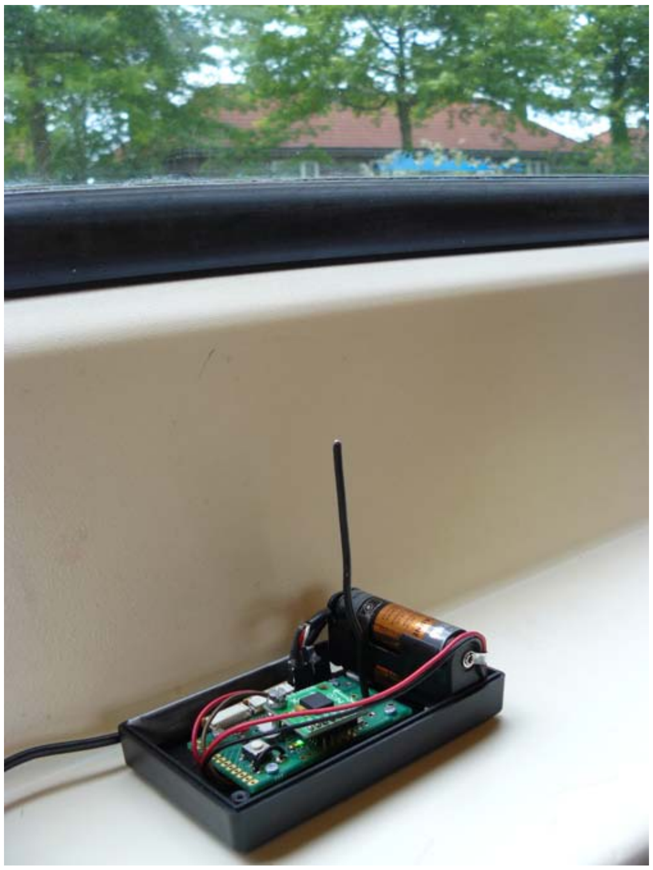
II. Data Collection and Analysis

As explained in the introduction, movement is used as discriminating feature. However, motion is multi-dimensional and too complex to use in wireless sensor nodes. Not only processing power and memory usage might be issues, also running complex algorithms for longer periods of time consumes large amount of energy, negatively influencing the operational time of the system. Two measures are taken to enable implementation on nodes:

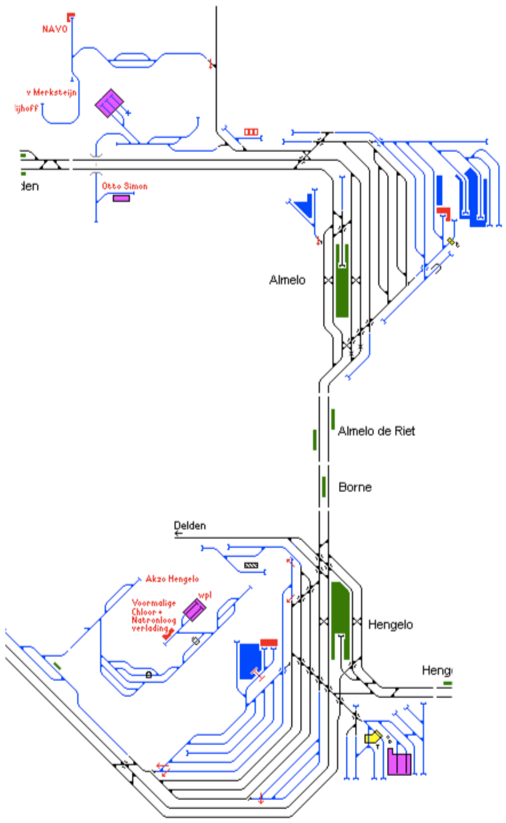
- limit the time the algorithm executes, and
- simplify the input data for the algorithm.

Under normal conditions a train’s composition will not change while underway and moving. If a change occurs under these circumstances, it will be accidentaly. The algorithm can be split into two phases:

- when the train starts moving, detect the initial set of carriages that belong to the train, and
- once the train is moving, only check those carriages in the initial set and ignore all others.



(a) Wireless sensor node



(b) Railway track

Figure 1. Data collection

The latter phase is accomplished by periodically pinging the initial set, which is a simple process that consumes less energy than is needed for the first phase.

A. Collecting the Data

To simplify phase 1, a train's movement is analysed to reveal those characteristics that are essential and those that can be safely ignored. Representative data sets are recorded on a track featuring multiple stops and curves (see Figure 1b), resulting in a wide variety of data. Figure 1a shows the wireless sensor node that is used to sample the data, consisting of an Ambient muNode 2.0 provisioned with an STMicroelectronics LIS3LV02DQ accelerometer. The maximum sample rate of this sensor is 640 Hz, but the used combination of hardware and software gives a maximum sample rate of 160 Hz. The sensor nodes are aligned with the horizontal x-axis in the driving direction, the y-axis in the horizontal sideways direction and the z-axis in the vertical direction.

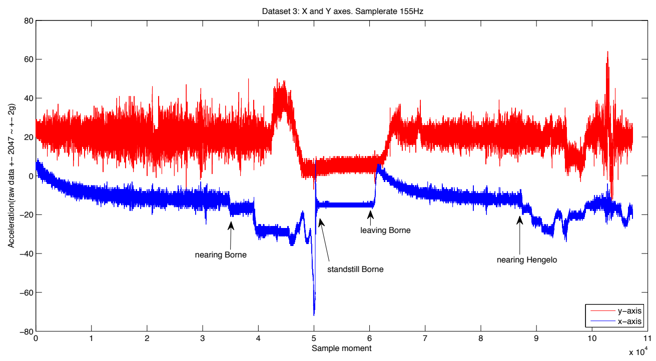
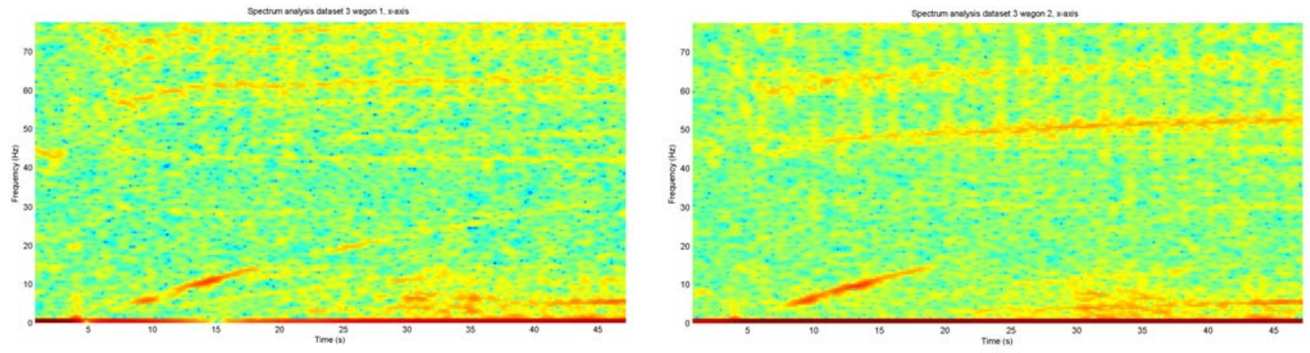


Figure 2. Accelerometer x- and y-axis

B. Analysing the Data

Figure 2 shows raw data with a sampling frequency of 155 Hz of a journey between two stations with one



(a) Spectrum carriage 1, train 1, x-axis

(b) Spectrum carriage 2, train 1, x-axis

Figure 3. Frequency spectrum of two carriages in the same train

station in between. The bottom graph shows the x-axis and halfway the train is standing still (no acceleration). The top graph depicts the sensor's y-axis and just before reaching the middle station the train is crossing a switch changing tracks. The data from both x- and y-axis (and, though not shown here, from the z-axis as well) contain a lot of noise and must be filtered with a high-off filter before it can be useful. Figure 3 shows the frequency spectrum for the x-axis data of two carriages in the same train starting to move. Only frequencies lower than the sampling frequency/2 are considered. The spectra are similar in the low frequencies, but quite different in the high frequencies. The spectra of two carriages in two different trains on the same part of the railway track (not shown) differ in all frequencies, but show similar characteristics in the lower frequencies. Closer inspection learns that a cut-off frequency of 2 Hz can be applied without losing discriminating features.

Data from the y- and z-axis of two carriages are similar irrespective whether they are in the same train or not and do not contain enough discriminating features. In the remainder of the paper only data from the x-axis of the accelerometer are considered.

To filter the data, a second order Butterworth low pass filter is used. This choice is made because this type of filter will run on the wireless sensor nodes. Figure 4 shows the result: the top and bottom graph are from carriages in the same train, while the middle one is in a different train. The graphs are synchronized, i.e., they are shifted in time so they show trains starting at the same time. In practise it will rarely happen that two trains in communication range will accelerate at exactly the same time. The data sampling rate in the original data set is 155 samples per second. Because the data is filtered at 2 Hz, this high sampling rate is overkill and could be reduced significantly in the final implementation. A frequency of 35 samples per second gives the same results as before. We did not test lower sampling rates.

C. *Data Correlation*

The last step in the algorithm is to check whether two carriages share the same context by way of correlating the data. The correlation process uses a sliding window over which the data is compared. A wider window normally leads to a more precise result, but also takes longer to produce this result. A smaller window gives a better reaction time, but the result is unreliable. So a trade-off has to be made. Figure 5 gives correlation results at a window size varying from 1 to 5 seconds for a time frame of 7000 samples or 45 seconds. Detection of the carriages is only active in phase 1 of the algorithm when the train starts moving, which is approximately during the first 2000 samples (phase 2 only pings known carriages). Therefore the results after 2000 samples will be ignored. For two carriages in the same train a window size of 155 samples or 1 second would already suffice. However for two carriages in different trains the window size should be at least 465 (3 seconds), or even better 620 samples or 4 seconds.

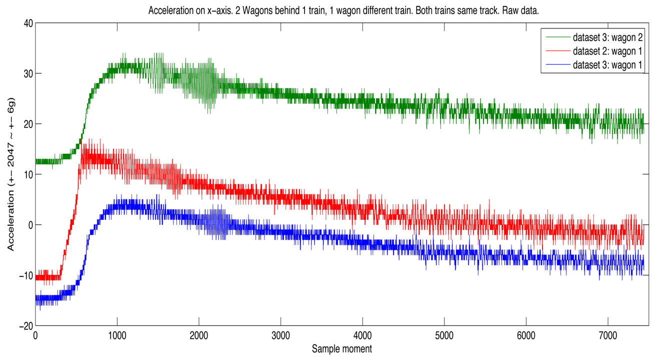
III. Implementation

In the following we discuss some implementation details.

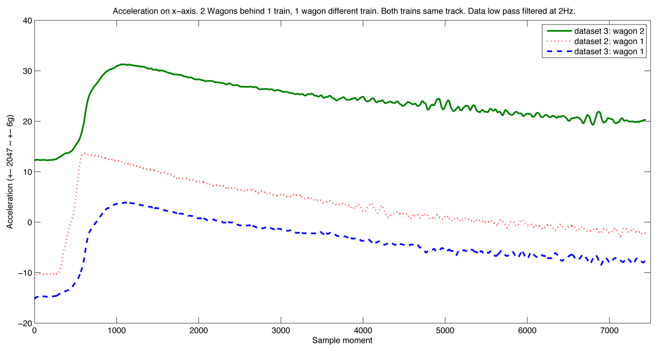
A. *Optimizations*

Matlab calculations, based on realistic data, in the previous section showed that to select a set of carriages that belong to the same train out of a larger population of carriages, movement, or more precise acceleration in the driving direction, can be used as a discriminating factor. However, the Matlab routines must be optimized or simplified before they will run on the wireless sensor nodes.

The Matlab program is centralized and assumes that all data is available when needed, which is not the case in the real world. The program and the data are distributed over the wireless nodes: each carriage must calculate its correlation with its neighbors and to do so it needs the movement information from its neighbors. This process is optimized by dynamically forming master/slave pairs. The

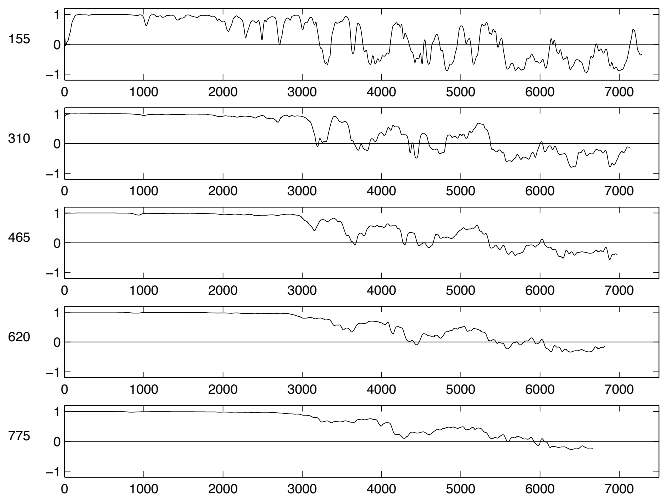


(a) Unfiltered data

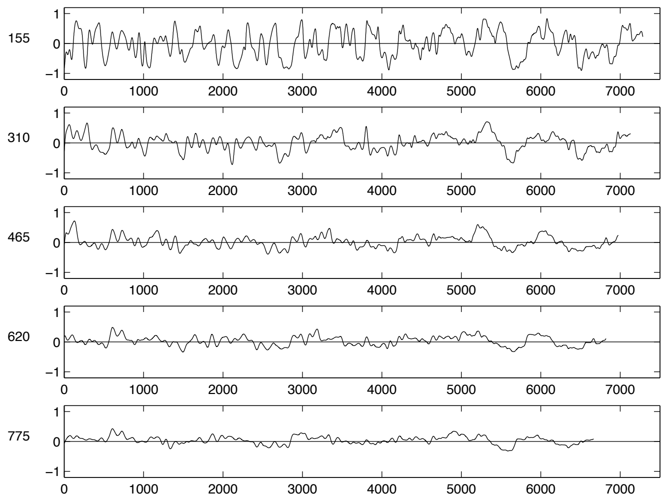


(b) Filtered data

Figure 4. X-axis acceleration data from three carriages



(a) two carriages in the same train



(b) two carriages in different trains

Figure 5. Correlation of two carriages with varying window sizes

slave sends its data to the master and after calculation the master sends the correlation results back to the slave. For every master/slave pair the movement data is communicated once and the correlation calculations is performed once, thus reducing both needed bandwidth and execution load. To balance the energy consumption on pairs the master and slave switch functionality after each correlation.

One more change in the original Matlab routines must be made before they can be implemented. The filter and correlation from the previous section are based on calculations that use floating point numbers. This puts too much of a burden on the sensor node and a fixed point calculation would be better, though at the cost of possible loss of precision. Errors in rounding results would accumulate and might lead to significant deviations over time. In Figure 6 the difference between floating point and fixed point correlation calculation is shown. The deviation starts to show after 15 to 20 seconds. Since the determination of the train composition takes place in the first 5 to 10 seconds, replacing floating point by fixed point calculations does not influence the end result.

B. Timing and Memory Usage

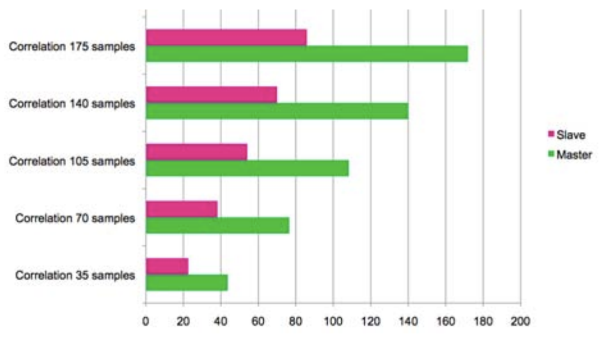


Figure 7. Execution times on a muNode with MSP430 microcontroller

In each master/slave pair, the correlation calculation is

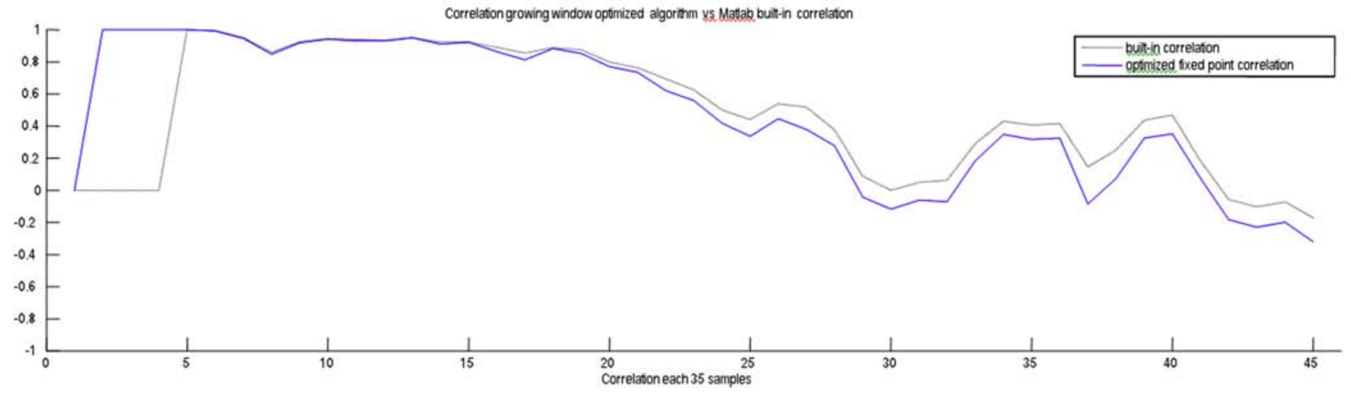
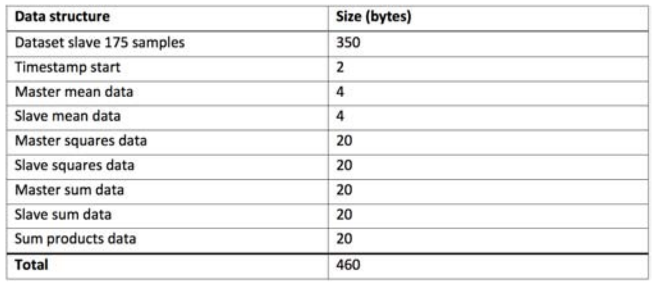


Figure 6. Comparison of floating point and fixed point calculations

done by the master while both partners filter their own data. In addition, the slave needs some time to assemble the packets to be send to the master. In Figure 7 the execution times for filtering and correlation for master and slave are shown. Assuming a correlation window size of 5 seconds (corresponding with 175 samples at 35 Hz), the time needed for the calculations is 171.9 ms and 85.9 ms for master and slave respectively. This leaves room for approximately 6 correlation calculations per second.



Data structure	Size (bytes)
Dataset slave 175 samples	350
Timestamp start	2
Master mean data	4
Slave mean data	4
Master squares data	20
Slave squares data	20
Master sum data	20
Slave sum data	20
Sum products data	20
Total	460

Figure 8. Memory consumption per master/slave pair

The memory consumption of our algorithm on each node depends on the number of master/slave pairs each node has formed with neighboring nodes. The minimum memory consumption for the correlation calculation algorithm on a node is (175+35)*2=420 bytes. Each node stores its own data samples for the given window as well as 35 extra samples, since the point at which master/slave pairs are formed is not the same for each pair. The table in Figure 8 summarizes the memory consumption per master/slave pair. The total amount of bytes is 460 per master/slave pair.

A standard muNode 2.0 has 10kB RAM available. For the normal operation of the node 2kB has been reserved, which leaves 8kB for the correlation algorithm. Given the memory consumption of 460 bytes for a participating node and the availability of 8192 bytes, a node can store up to 16 master/slave pairs in memory.

IV. Conclusion

In this paper we have presented an opportunistic sensor system that makes a selection out of a much larger set based on a common context. In this case, motion information is used to distinguish carriages belonging to different trains. The motion information consists of data obtained by 3-dimensional accelerometers attached to individual train carriages. This data stream then was analyzed with Matlab on a PC to extract the best possible features to discriminate carriages. While sideways motion gives information on track changes and vertical movement indicates the quality of a track, the best information for our purpose is movement in the direction of travelling (x-axis). A spectrum analysis learns that not all frequency components in the sampling data are equally useful: only those below a frequency of around 2 Hz are significant to separate carriages. After filtering, the data from two different carriages are correlated with a correlation window of 5 seconds. We found that a smaller window of approximately 1 second suffices to find two carriages in the same train, but also leads to false positives for carriages in different trains. Extending the window to 5 seconds, no false positives were detected, but this needs further analysis.

After this theoretical confirmation that motion information can be used for context awareness, the algorithm is implemented on wireless sensor nodes. However, the nodes used have several limitations. One limitation is the absence of floating point calculations. First of all the filter and correlation routines are rewritten, so only fixed point calculations are used. This might lead to errors due to accumulation of rounding successive results, but we showed that in the time frame the algorithms run this is not a problem.

The second limitation is power consumption. Filtering and correlation are so computing intensive that they cannot run over longer periods of time without exhausting the battery quickly. A first step is the reduction of the sampling rate from the original 160 Hz to 35 Hz. This is made possible because only the lower frequencies in the sampling data are significant. A lower sampling frequency would have been possible, but this does not substancially contribute to decreasing the processing load. Sampling data, filtering and performing one correlation per second takes 171.9 ms (worst case), which results in a duty cycle of around 17 percent. This exhausts the battery in a couple of hours, at most days, where 6 months is needed. The solution is found by executing the algorithm only for a period of 10 seconds when the train starts moving after each stop. This is enough to establish which carriages belong to one and the same train. During the ride, the initially detected carriages (but not the sequence of the carriages) needs to be confirmed, which can be accomplished by pinging all known carriages at regular intervals.

The last limitation is memory capacity. In our algorithm carriages are correlated in pairs. A carriage is part of as many pairs as it has neighbors. Each pair consumes up to 460 bytes of memory in both partners. With the given memory capacity, a node can accommodate up to 17 neighbors. With a maximum of 6 correlations per second it takes a node 3 seconds to check all its neighbours.

The circumstances in which the data is collected and the implementation is tested form a worst case scenario. The trains that are used in the tests are of the same type with similar characteristics. They all exhibit the same pattern in acceleration and braking, making the data to correlate very similar. We suspect this is the main reason it takes up to 5 seconds to check carriages from different trains (and only 1 second when they are in the same train). This needs further investigation.

Another future research topic is the use of better accelerometers. Those used now measure up to 2g and can be used on trains that accelerate moderately. However, heavy trains that accelerate and brake more slowly have less distinctive movement patterns and need more sensitive sensors.

ACKNOWLEDGMENT

This work is supported by the iLAND Project, ARTEMIS Joint Undertaking Call for proposals ARTEMIS-2008-1, Project contract no. 100026

REFERENCES

[1] H. Scholten, R. Westenberg, and M. Schoemaker, *Sensing train integrity*, IEEE Sensors 2009 Conference, pages 669674, Los Alamitos, October 2009. IEEE Computer Society Press.

[2] Schwartz, R.S. and van Eenennaam, E.M. and Karagiannis, G. and Heijenk, G.J. and Klein Wolterink, W. and Scholten, J, *Using V2V communication to create Over-the-horizon Awareness in multiple-lane highway scenarios*, IEEE Intelligent Vehicles Symposium (IV) 2010, 21-24 June 2010, La Jolla, CA, USA. pp. 998-1005. IEEE Computer Society Press. ISSN 1931-0587 ISBN 978-1-4244-7866-8

[3] M. Kumar, *Distributed computing in opportunistic environments*, UIC 09: Proceedings of the 6th International Conference on Ubiquitous Intelligence and Computing, pages 11, Berlin, Heidelberg, 2009. Springer-Verlag.

[4] L. Lilien, A. Gupta, and Z. Yang, *Opportunistic networks for emergency applications and their standard implementation framework*, Performance, Computing, and Communications Conference, 2002. 21st IEEE International, 0:588593, 2007.

[5] L. Pelusi, A. Passarella, and M. Conti, *Opportunistic networking: data forwarding in disconnected mobile ad hoc networks*, Communications Magazine, IEEE, 44(11):134 141, november 2006.

[6] R. Murty, G. Mainland, I. Rose, A. R. Chowdhury, A. Gosain, J. Bers, and M. Welsh, *Citysense: A vision for an urban-scale wireless networking testbed*, Proceedings of the 2008 IEEE International Conference on Technologies for Homeland Security, pages 583588. IEEE Press, 2008.

[7] M. Wirz, D. Roggen, and G. Troster, *Decentralized detection of group formations from wearable acceleration sensors*, Proceedings of the 2009 IEEE International Conference on Social Computing, page IEEE Press, Aug. 2009.

[8] M. Wirz, D. Roggen, and G. Troster, *A methodology towards the detection of collective behavior patterns by means of body-worn sensors*, Proc. of UbiLarge workshop at Pervasive, 2010.

[9] N. Davies, D. P. Siewiorek, and R. Sukthankar, *Special issue: Activity-based computing*, IEEE Pervasive Computing, 7(2):2021, 2008.

[10] S. Mann, *Humanistic computing: wearcom as a new framework and application for intelligent signal processing*, Proceedings of the IEEE, 86(11):21232151, 1998.

[11] B. Myers, J. Hollan, I. Cruz, S. Bryson, D. Bulterman, T. Catarci, W. Citrin, E. Glinert, J. Grudin, and Y. Ioannidis, *Strategic directions in human-computer interaction*, ACM Computing Surveys, 28(4):794809, 1996.

[12] S. Bosch, M. Marin-Perianu, R.S. Marin-Perianu, J. Scholten and P.J.M. Havinga, *FollowMe! Mobile Team Coordination in Wireless Sensor and Actuator Networks*, Proceedings of the IEEE International Conference on Pervasive Computing and Communications 2009, 9-13 March 2009, Galveston, Texas, USA. pp. 151-161. IEEE Computer Society Press. ISBN 978-1-4244-3304-9

[13] R.S. Marin-Perianu, C. Lombriser, P.J.M. Havinga, J. Scholten and G. Troster, *Tandem: A Context-Aware Method for Spontaneous Clustering of Dynamic Wireless Sensor Nodes*, Proceedings of the First International Conference on Internet of Things (IOT2008), March 2008, Zurich, Switzerland. pp. 341-359. Lecture Notes in Computer Science (4952). Springer Verlag. ISBN 978-3-540-78730-3.

[14] Scholten, J. and Westenberg, R. and Schoemaker, M. , *Trainspotting, a WSN-based train integrity system*, The Eighth International Conference on Networks, ICN 2009, 1-6 March 2009, Gosier, France. pp. 226-231. IEEE Computer Society Press. ISBN 978-0-7695-3552-4.