# INTEGRATING IEEE 802.15.4 – BASED SENSOR NETWORKS IN INDUSTRIAL PROCESS CONTROL

***Emanuele Toscano, Mario Collotta, Salvo Vittorio***

Department of Computer Engineering and Telecommunications
University of Catania,
Italy
etoscano@diit.unict.it, mcollotta@diit.unict.it, svittorio@diit.unict.it

**Abstract**

This article explains and analyses a technique whose goal is the integration of control process technologies with the functionalities of a WSN (Wireless Sensor Network) based on IEEE 802.15.4/ZigBee. We describe a set of sample scenarios in industrial applications where we could use WSN and real time to monitor critical processes. Then, a system design and implementation overview is developed that covers our network architecture and some design and implementation issues.

Index Terms: Process Control, RTAI, ZigBee, IEEE 802.15.4, Wireless Sensor Networks.

## I. INTRODUCTION

In embedded industrial automation and process control applications increasing use is being made of wireless communication devices instead of, or combined with, traditional wired connections. Wireless systems are in great demand due to their flexibility. Devices can be rapidly installed without the added cost and time required by the installation of cables. Wireless technology is even more advantageous if installations are temporary. Of the various systems available, 802.11 [1] and Bluetooth (BT) [2 – 4] seem particularly suitable for process control applications. Growing interest has recently been shown in ZigBee [5] or IEEE 802.15.4 [6, 7] for use in embedded applications requiring low data rates and low power consumption. Process control systems represents a class of real-time applications that provide large benefits for industrial companies. Many of the attempts aim at eliminating cabling in the industry environment and replace them with low cost wireless technologies that need less maintenance and provide a better power management but require more software complexity. Sensors are essential to Industrial Automation (IA) providing a link between control systems and the physical world where they are applied. New hardware and software for control systems are creating many new possibilities for automation but cost-efficient use of sensors is restricted in industrial applications by:

– the expense of wiring and maintaining sensor networks,

– the safety and regulatory obstacles to running cables in constricted or dangerous areas,

– protocol incompatibility between various sensor types,

– the high cost of installation,

– the high failure rate of connectors.

These are just some of the factors driving users to investigate wireless alternatives. For the industrial applications, the systems must require operations with minimum operator intervention. For a wireless network, this takes to a system that is self-organizing and self-configuring.

Also Real Time techniques are widely used in industrial production area especially for control process. The activities that mainly need a real time elaboration are: regulation and control of chemical and nuclear factories, control of complex production systems, industrial automation, industrial monitoring systems. Goal of Process Control is to check continuously all the steps of the production. In some of them the attention is mostly concentrated on one parameter: the *criticality***.** For example in manufacturing or petrochemical automation the first issue that has to be guaranteed is the security for all people who are working there.

There are also some particular steps that need a continuous checking. In these steps Real Time techniques have to be used to guarantee security. For Example the chemic reactions of some liquid need a real time monitoring to check the toxic level and some special communication techniques to avoid the block of the chemic processes because of external events.

Three points are essential in these cases:

– accuracy of the process monitoring;

– stability in all steps of the process;

– reactivity of the system to particular events.

In these three points software and hardware technologies play an essential role. In fact all the automation company every year invests a lot of money in hardware and software solutions to minimize the total cost of the production and maximize profit and safety. However, there are cases where more practical and cost-effective network deployment could be achieved by wireless networking, i.e., when data is less critical. For example, a pervasive Wireless Sensor Network could be deployed over and beyond the field, so that different hard real-time control cells could be connected among them and also with a set of shared external sensors.

Our aim is to develop an application where we can gradually integrate the wireless communication potentialities in the context of real-time process control.

## II. APPLICATION OVERVIEW

Goal of this paper is to realize an application that integrate IEEE 802.15.4 –based Sensor Networks in Process Control Automation Systems to handle asynchronous events that does not require hard real-time guarantees, as well as to enable affordable data dissemination. A similar approach can lead to multiple benefit, as it would be possible for each cell to access global parameters thanks to local sink nodes deployed in each cell (Fig. 1). These parameters may be external or internal, e.g., winds, tides or seismic events, as well as status of machines, warehouses or plants. These parameters will be all available with extremely low cost compared with current installations, they can be transmitted to the requesting devices or collected for historical analysis.

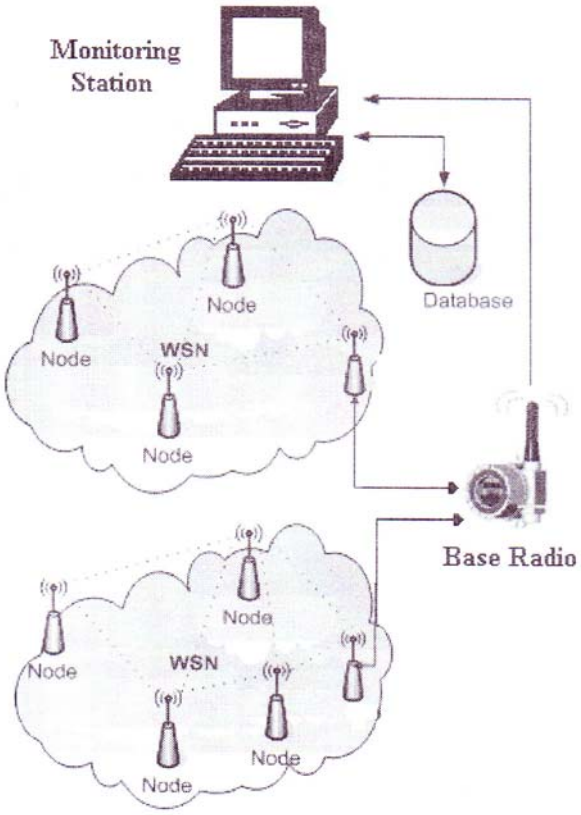


Fig. 1 WSN Architecture

Currently the only available standard for control networks is ZigBee [5]. Besides, it also requires a low power consumption and cost. ZigBee's current focus is to define a general-purpose, inexpensive, self-organizing mesh network that can be used for industrial control, embedded sensing, medical data collection, building automation, home automation, etc. The resulting network will use very small amounts of power so individual devices might run for a year or two using the originally installed battery. However, critical parts of process control may still need hard real-time networks and operating systems for maximum reliability and timeliness guarantee. Thus, our solution is to integrate WSN model of data dissemination [13, 14] with hard real-time process control in a two tiered solution. The first tier is made up of the hard real-time control system that consecutively executes a control loop, while the second tier is the less critical but pervasive ZigBee network. As WSN nodes may notify changed operating conditions to which the control has to react, low response time are desirable, but without interference on hard real-time control. For this reason data acquisition from ZigBee motes is scheduled within the hard real-time operating system.

## III. SIMPLIFIED MODEL

This paper shows our implementation of a simplified model of WSN integrated in process control. We decided to divide our application in two parts:

- Hard Real Time?
- Soft Real Time.

The former part has in charge to handle processes that need time guarantee with a Real-Time scheduling protocol. In particular we used the Earliest Deadline First (EDF) [8] algorithm, that has been proved to be an optimal scheduling algorithm for uniprocessor systems [9]. That is, if a collection of independent jobs, each characterized by an arrival time, an execution requirement, and a deadline, can be scheduled (by any algorithm) such that all the jobs complete by their deadlines, the EDF will schedule this collection of jobs such that they all complete by their deadlines. So, compared to fixed priority scheduling techniques like rate-monotonic scheduling, EDF can guarantee all the deadlines in the system at higher loading. With scheduling periodic processes that have deadlines equal to their periods, EDF has a utilization bound of 100%. That is, EDF can guarantee that all deadlines are met provided that the total CPU utilization is not more than 100%.

Thus we created a set of high real-time tasks that emulate a typical process control loop.

We decided to implement this part using RTAI (Real-Time Application Interface) Linux [10], developed by Politecnico di Milano. This library is native to work in kernel mode and it is in a experimental level. Including this library on linux bring us to a new version of linux kernel (Real-Time). RTAI introduces a new level called RTHAL (Real Time Hardware Abstraction Layer) between hardware and kernel linux. RTHAL includes all critic functions in an unique structure to be able to handle them according to the real time scheduling policy. RTHAL has the role to intercept the system call and redirect them to the function that it includes (Fig. 2). If RTAI is not ACTIVE we will have the normal linux kernel, instead if RTAI is ACTIVE only real time functions can access to the hardware. In this case Linux is handled as a normal process with low priority.

RTAI extends the functions of the normal Linux Kernel, but it does not need to be loaded at system boot, as RTAI is composed by some kernel modules that could be loaded using linux command "insmod". RTAI can also be used in User Space mode, through the LXRT library. In User Space the portability of the real time processes can be improved, as in this level they are independent from the linux kernel in which they were realized.

The second tier of our system is based on ZigBee technologies to exchange control information. We adopted the RF ZigBee modules provided by Maxstream (now Digi), called XBee [11].

This level is developed to handle some aperiodic events during the real time processes execution. Let's consider a factory divided in working cells. Each room has its sensors network that monitors the critical processes in the cell and also there could be a shared WSN to have a global view of the entire environment. Local data can be disseminated thanks to the shared WSN. This could be useful to communicate and exchange data when aperiodic events happen in cells or to periodically transmit the status of equipments.

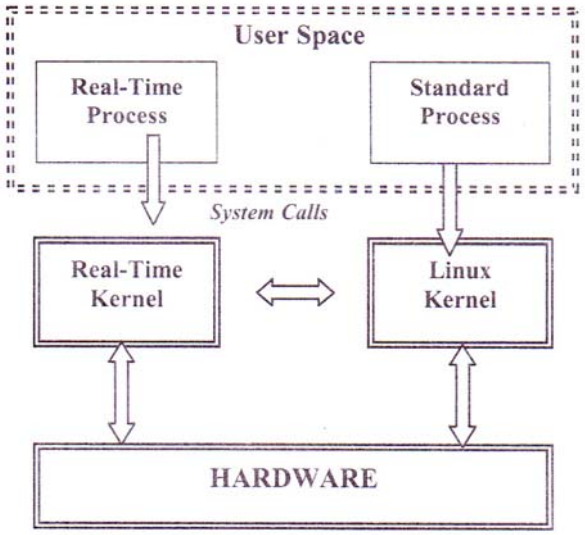


Fig. 2 RTAI Architecture

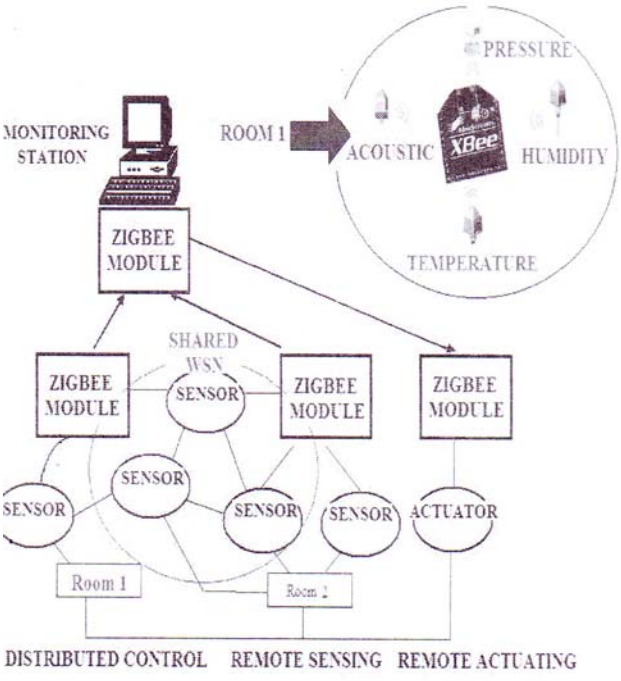


Fig. 3 Industrial scenario

Possible aperiodic events could be:

- heartquake,
- voltage spike,
- fire alarms,
- gas/liquids outflow.

Once the monitor station receives these alarms from the ZigBee modules set as trasmitters (connected directly either to field sensors or to the shared Sensor Network), it sends a signal to the appropriate actuators that will handle the situation; for example it could block the machines present in the room where the dangerous events happened.

RF modules belonging to a cell can be either Transmitter or Receiver and create a star-shaped network where modules used as Transmitter are referred to some aperiodic event (Fig. 4); for example we could associate to them some sensors that give data when some aperiodic events happen. In order to save energy, several sleep modes can be used, that enable the RF module to enter states of low-power consumption when not in use. In this case the ZigBee module that was in sleep mode resume and send a packet to the centralized RF Module set as Receiver

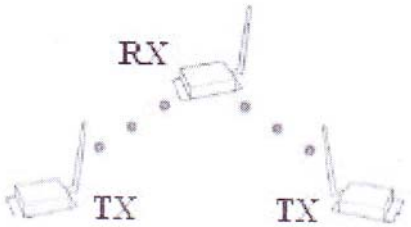


Fig. 4 Transmission scheme

The Receiver could be associated, in the example described above, to one sensor to measure the current state of the toxic level and it can apply the right actions based on the current value of that sensor. Receiver's action may depend on the current state of the critic process.

For example if the presence of some people is detected near the room where the chemic – toxic reactions are, the receiver would receive that event from the RF Module connected to the door and if the current state of the monitored critic process is in a level dangerous for people (e.g. if concentration of toxic substance is upon a threshold) it will fire an alarm and possibly do an action such as the block of the door.

Besides data transmissions, the second tier implements also ZigBee modules configuration of the most important paramteres, such as:

- dynamic number of transmitters,
- the address of transmitters and receiver,
- the radio channel where transmitters and receiver have to work. It depend on the ZigBee chip used *(0x0C - 0x17* for XBee-PRO, *0x0B – 0x1A* for XBee).

We used for our application ZigBee Pro chip for the Transmitters and ZigBee chip for the Receiver, and we set a common channel for their communication.

## IV. CODE PROGRAMMING

To implement our model we decided to use RTAI for the Hard Real-Time part and the API mode of XBee ZigBee modules [12] for the Soft part, as described in the previous

paragraph. Communication with XBee modules is implemented as an hard real-time task working in user mode. We created a configuration file where we can modify the scenario of our environment. We implemented several hard-real time tasks that model critical process control (tier one). We also decided to implement communication part (tier two) as a periodic task in the Real Time part of our application to be able to handle better some parameters in user mode, as RTAI was developed to work in Kernel mode. Fig. 5 is an example of LXRT code:

```
task = rt_task_init(nam2num("Name"), 1, 0, 0);
mlockall(MCL_CURRENT | MCL_FUTURE);
start_rt_timer(nano2count(PERIOD_NS));
rt_make_hard_real_time();
rt_task_make_periodic(task, now + nano2count(PERIOD_NS),
                      nano2count(PERIOD_NS));

for(i = 1; i <= 30; i++) {
  rt_task_wait_period();
  rtai_print_to_screen("Inizio job %d\n", i);
  rt_busy_sleep(SEC_IN_NS);
  rtai_print_to_screen("Fine job %d\n", i);
}

rt_make_soft_real_time();
stop_rt_timer();
munlockall();
rt_task_delete(task);

return 0;
}
```

Real time code

Fig. 5 Example LXRT-RTAI code

LXRT is the library, included in RTAI, for the User mode setup ("rtai_LXRT.h").

To work well a LXRT task has to realize the following steps:

– Set the scheduling to FIFO policy that let us to have better performances.

– Create a link to the agent that deal with real time. In such way the LXRT delegate this agent for the real time system calls.

– Disable the RAM paging before to enter in hard real time mode (by calling a proper function), because the real time process can't be stopped from accesses to the memory. In this way code and data of processes are maintained in RAM.

– Activate the hard real time mode in which the process is removed from the linux running-queue and scheduled by RTAI scheduler. The normal linux processes are executed only if the RTAI scheduler has no task in execution.

– Execute the real time code.

– Go back to the soft real time mode (by calling a proper function) where the process is inserted back to the linux ready-queue and considered as a normal linux process.

– Delete the agent before to finish the execution to free the memory and to avoid faults, instability and inconsistency.

In the first step we need to create a set of tasks that have to work simultaneously, so we forked the father process.

*for (i = 0; i <NTASK-1; i++) {*
*if (!fork())*
*mytask_name = i*

Here *NTASK* is the task number that we want to use and the variable *mytask_name* is a integer that we need for the init function in RTAI to give an unique name to it. The next function so is the creation of RTAI task in User mode:

*RT_TASK* rt_task_init (int name, int priority, int stack_size, int max_msg_size)*

This function creates the real time agent as extention of the calling process. The priority defined in the function is referred to the execution handled by the RTAI scheduler. In our implementation however this parameter is not too relevant, as we set the priority by the value of process Deadlines. RTAI Init function so extends the Linux task structure, making it possible to use RTAI APIs that wants to access RTAI scheduler services. It needs no task function as none is used, but it does need to setup an RTAI task structure and initialize it appropriately as the provided services are carried out as if the Linux process has become an RTAI task also. Because of that it requires less arguments and returns the pointer to the RTAI task extension that is to be used in related calls.

After the creation of RTAI agent we need to switch in hard real time mode by calling the following function:

*rt_make_hard_real_time ();*

By calling *rt_make_hard_real_time()* a task suspends itself so that another schedulable Linux objects is switch in (full preemption). As soon as the new task is switched in, the RTAI task switching is called, without even exiting the just called Linux "schedule" function and the RTAI tasks is resumed in real time hardened mode. When it has nothing to else do, it will call RTAI *reschedule* and such a function will schedule a Linux object again. Notice that, full interoperability of Linux/RTAI context switches is assured by a common context switch function available in Linux *sched.c.*

Subsequentely we use:

*rt_task_make_periodic(mytask, time, period);*

Here *mytask* is the result of *rt_task_init(); rt_task_make_periodic* makes a task run periodically by marking the task, previously created with rt_task_init(), as suitable for a periodic execution, with a period period, when *rt_task_wait_period()* is called.

The time of first execution is defined through *startjime* function *{start_rt_timer* previously declared) that is an absolute value measured in clock ticks. The timer is the main step to allow having deterministic timing constraint inside the RTAI created task. The timer can be started or stopped with API defined in *"rta_sched.h".* At this point to implement EDF scheduling RTAI provides a function called:

*rt_task_set_resume_end_time (resume, end);*

It set the absolute instant of resume and deadline of a RTAI process. This process is descheduled and when it resume it is inserted into the scheduler queue according to the deadline. If we specify negative parameters it is possible to set a resume related to the previous one and a new deadline related to the actual resume value.

After this step start the real time code. To emulate the examples previously discussed we wrote a function as real time code that take the CPU and work on it.

```
for (i. = 1; i <= cycles; i++) { printf("Init job \n");
for(j = 1; j <=cydes_1; j++) {
printf("CPU work\n"); CPU_Emulation (mytask_name);
   }
  }
void CPU_Emulation () {
printf("INIT TASK %d \n",mytask_name);
for(j = 1; j <=cydes_l; j++) CPU_operation;
printf("END TASK %d \n",mytask_name);
rt_task_set_resume_end_times(-resume, -
        deadline);
         }
```

Once that real time processes had finished their work we need to stop the timer that sets the timer back into its default mode (periodic). Finally when a task is exiting or hard real time is not needed any more we should call the *rtai_make_soft_real_time()* function following it with *rt_task_delete()* that is used to detach RTAI from the Linux task structure.

This API, in User space, can also be "forgotten" and RTAI will clean everything by itself, but it is strongly suggested to delete a task inside your own code to avoid any side effect.

```
rt_task_delete(mytask_name);
```

The following image gives the idea of our EDF scheduling described above.

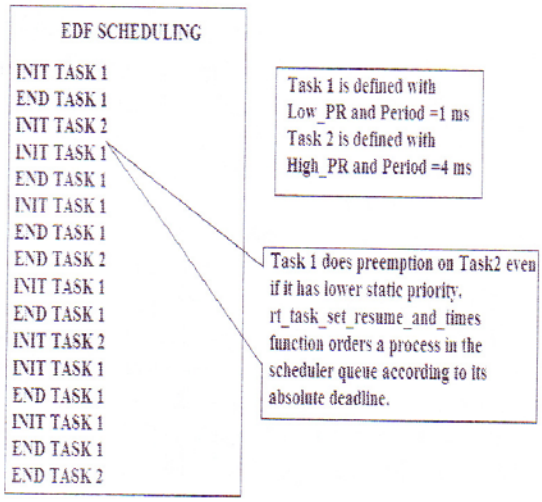


Fig. 6 RTAI EDF Scheduling

The soft real part of our application is based on ZigBee communication as described previously/ In our scenario implementation, supposed an aperiodic event occurs, the RF ZigBee module (Transmitter) wake up and it will send a packet data to the another RF module set as receiver.

For the serial communication we implemented a set of C functions that communicate through RS-232 serial connection using the built-in API protocol of the XBee modules. The most important functions are presented below:

- *xbee_sleep*, puts a node in deep sleep state;
- *xbee_wake*, wakes up a node from sleep;
- *xbee_AT_get_param*, reads a configuration parameter from a XBee module;
- *xbee_AT_setj>aram*, sets a configuration parameter into a XBee module;
- *xbee_transmit_16*, transmits a data frame using short 16-bit addressing format;
- *int xbee_receive_apijrame*, fetch and decode a received API frame.

Notice that also the xbee AT set and get function actually do not use AT commands, but API frames that wraps the same AT commands described in the reference manual of XBee modules that can be used into AT mode. For example:

```
xbee_AT_set_param (port, "MY", adr, size (adr))
```

calls the "MY" command, that set the address of a XBee module.

The receiver never goes in sleep because it has to monitor the hard real time processes and it has to react if transmitters will send data. Based on the source, the message type and the current state of the hard real time processes receiver will decide the correspondent action; for example it could be not to open the door and display "Access Denied" to whom wanted to enter the room when high concentration of toxic substances is detected. For our emulation, the decision was based on a simple *if-else* statement:

```
If(j equals mytask_name in execution) Do_Op_A();
//function that emulate
        //the critic action
else
Do_OP_B () ;
```

Either the scheduling part or the ZigBee part was tested on a single Laptop dual core using Ubuntu 7.10 Gutsy and RTAI 3.6testl.

The scenario developed was composed of 2 RT TASK with EDF Scheduling; 2 TX and 1 RX using RF ZigBee modules. The response of RTAI suffers of some bug solved case by case when they occur and it is difficult to work with it because most of the documentation about functions, kernel and user is still not completely available.

## V. CONCLUSION

The design and emulation of a wireless sensor platform targeted for process control systems was discussed and presented. The environment presented is a semplified version of a real industrial application, but our

application could be easily extended. Extended testing of the integration model lets us believe its applicability in a real industrial environment reducing the global cost of the factory (self-mantaining, self-configuration) and maximizing the profit.

## REFERENCES


[1] IEEE 802.11, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, Standard, IEEE, Aug. 1999.

[2] Bluetooth SIG. Specification of the Bluetooth System - Version LIB, Specification Vol. 1 & 2, Feb. 2001.

[3] Bluetooth SIG, "Bluetooth Core Specification", Nov. 004.

[4] M. Collotta, L Lo Bello, O. Mirabella, "Deadline-Aware Scheduling Policies for Bluetooth Networks in Industrial Communications". In Proceedings of the IEEE Second International Symposium on Industrial Embedded Systems - SLES'2007, Lisbon, Portugal, 4-6 July 2007, pagesl56-163, Digital Object Identifier 10.1109/SLES.2007.4297330.

[5] ZigBee Alliance, http://www.zigbee.org.

[6] 802.15.4 - 2003 IEEE Standard for Information Technology-Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low- Rate Wireless Personal Area Networks (LR-WPANs), IEEE, Oct. 2003.

[7] 802.15.4 - 2006 LEEE Standard for Information technology-Telecommunications and information exchange between systems— Local and metropolitan area networks— Specific requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless. LEEE, Sept. 2006.

[8] C. L. Liu and J. W. Layland. "Scheduling Algoritms for Multiprogramming in a Hard-Real-Time Environment", Journal of ACM, Vol. 20, No. 1, January 1973.

[9] Michael L. Dertouzos. Control robotics: the procedural control of physical processes. In Proceedings of IMP Congress, pp. 807-813, 1974.

[10] RTAI Official Website, available at https://www .rtai.org/

[11] Maxstream Official Website, available at http://www.maxstream.net/

[12] Maxstream XBee/XBee Pro product manual, available at http://ftp 1 .digi.com/support/documentation/manual xboem-rf-modules_802.15.4_v1.xAx.pdf

[13] I. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, A survey on sensor networks, IEEE Commun. Mag. 40 (8) (2002) 102-114.

[14] D. Estrin, L. Girod, G. Pottie, and M. Srivastava. Instrumenting the world with wireless sensor networks. In International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2001), Salt Lake City, Utah, May 2001