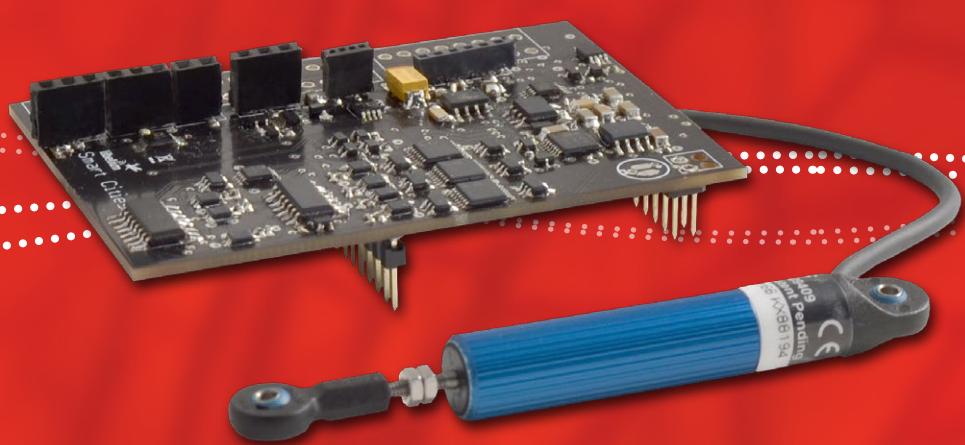


# Smart Cities

## Technical Guide



Document version: v4.2 - 04/2013  
© Libelium Comunicaciones Distribuidas S.L.

# INDEX

<b>1. General .....</b>	<b>4</b>
1.1. General and safety information .....	4
1.2. Conditions of use .....	4
<b>2. WaspMote Plug &amp; Sense!.....</b>	<b>5</b>
2.1. Features .....	5
2.2. Sensor Probes .....	5
2.3. Solar Powered .....	6
2.4. Programming the Nodes .....	7
2.5. Radio Interfaces .....	8
2.6. Program in minutes .....	9
2.7. Data to the Cloud .....	9
2.8. Meshlium Storage Options .....	10
2.9. Meshlium Connection Options .....	10
2.10. Models .....	11
2.10.1. Smart Cities .....	12
<b>3. Hardware .....</b>	<b>14</b>
3.1. General Description .....	14
3.2. Specifications .....	14
3.3. Electrical Characteristics .....	14
<b>4. Sensors .....</b>	<b>15</b>
4.1. Particle Sensor (PM-10) – Dust Sensor (GP2Y1010AU0F) .....	15
4.1.1. Specifications .....	15
4.1.2. Measurement Process .....	15
4.1.3. Socket .....	17
4.2. Crack detection sensors (Vishay) .....	18
4.2.1. Specifications .....	18
4.2.2. Measurement Process .....	19
4.2.3. Socket .....	20
4.3. Crack propagation sensors (Vishay) .....	21
4.3.1. Specifications .....	21
4.3.2. Measurement Process .....	22
4.3.3. Socket .....	23
4.4. Linear Displacement Sensor - Crack measurement (SLS095) .....	24
4.4.1. Specifications .....	24
4.4.2. Measurement Process .....	24
4.4.3. Socket .....	26

4.5. Noise Sensor (Microphone WM-61A).....	27
4.5.1. Specifications.....	27
4.5.2. Measurement Process.....	27
4.5.3. Socket.....	29
4.6. Ultrasonic Sensor (MaxSonar® from MaxBotix™) .....	29
4.6.1. Specifications.....	29
4.6.2. Measurement Process.....	31
4.6.3. Socket.....	33
4.7. Humidity Sensor (808H5V5).....	34
4.7.1. Specifications.....	34
4.7.2. Measurement Process.....	34
4.7.3. Socket.....	35
4.8. Temperature Sensor (MCP9700A) .....	35
4.8.1. Specifications.....	35
4.8.2. Measurement Process.....	35
4.8.3. Socket.....	36
4.9. Luminosity Sensor (LDR) .....	37
4.9.1. Specifications.....	37
4.9.2. Measurement Process.....	37
4.9.3. Socket.....	38
4.10. Sensor interruptions.....	38
4.11. Sockets for casing.....	39
<b>5. Board configuration and programming .....</b>	<b>41</b>
5.1. Hardware configuration .....	41
5.2. API.....	41
<b>6. Consumption .....</b>	<b>46</b>
6.1. Power control .....	46
6.2. Tables of consumption .....	46
6.3. Low consumption mode .....	47
<b>7. API Changelog .....</b>	<b>48</b>
<b>8. Documentation changelog .....</b>	<b>49</b>
<b>9. Maintenance .....</b>	<b>50</b>
<b>10. Disposal and recycling .....</b>	<b>51</b>

# 1. General

## 1.1. General and safety information

- In this section, the term "WaspMote" encompasses both the WaspMote device itself and its modules and sensor boards.
- Read through the document "General Conditions of Libelium Sale and Use".
- Do not allow contact of metallic objects with the electronic part to avoid injuries and burns.
- NEVER submerge the device in any liquid.
- Keep the device in a dry place and away from any liquid which may spill.
- WaspMote consists of highly sensitive electronics which is accessible to the exterior, handle with great care and avoid bangs or hard brushing against surfaces.
- Check the product specifications section for the maximum allowed power voltage and amperage range and consequently always use a current transformer and a battery which works within that range. Libelium is only responsible for the correct operation of the device with the batteries, power supplies and chargers which it supplies.
- Keep the device within the specified range of temperatures in the specifications section.
- Do not connect or power the device with damaged cables or batteries.
- Place the device in a place only accessible to maintenance personnel (a restricted area).
- Keep children away from the device in all circumstances.
- If there is an electrical failure, disconnect the main switch immediately and disconnect that battery or any other power supply that is being used.
- If using a car lighter as a power supply, be sure to respect the voltage and current data specified in the "Power Supplies" section.
- If using a battery in combination or not with a solar panel as a power supply, be sure to use the voltage and current data specified in the "Power supplies" section.
- If a software or hardware failure occurs, consult the Libelium Web **Development section**.
- Check that the frequency and power of the communication radio modules together with the integrated antennas are allowed in the area where you want to use the device.
- WaspMote is a device to be integrated in a casing so that it is protected from environmental conditions such as light, dust, humidity or sudden changes in temperature. The board supplied "as is" is not recommended for a final installation as the electronic components are open to the air and may be damaged.

## 1.2. Conditions of use

- Read the "General and Safety Information" section carefully and keep the manual for future consultation.
- Use WaspMote in accordance with the electrical specifications and the environment described in the "Electrical Data" section of this manual.
- WaspMote and its components and modules are supplied as electronic boards to be integrated within a final product. This product must contain an enclosure to protect it from dust, humidity and other environmental interactions. In the event of outside use, this enclosure must be rated at least IP-65.
- Do not place WaspMote in contact with metallic surfaces; they could cause short-circuits which will permanently damage it.

Further information you may need can be found at: <http://www.libelium.com/development/waspmote>

The "General Conditions of Libelium Sale and Use" document can be found at:  
[http://www.libelium.com/development/waspmote/technical\\_service](http://www.libelium.com/development/waspmote/technical_service)

## 2. Waspmote Plug & Sense!

The new Waspmote Plug & Sense! line allows you to easily deploy wireless sensor networks in an easy and scalable way ensuring minimum maintenance costs. The new platform consists of a robust waterproof enclosure with specific external sockets to connect the sensors, the solar panel, the antenna and even the USB cable in order to reprogram the node. It has been specially designed to be scalable, easy to deploy and maintain.

**Note:** For a complete reference guide download the "Waspmote Plug & Sense! Technical Guide" in the [Development section](#) of the [Libelium website](#).

### 2.1. Features

- Robust waterproof IP65 enclosure
- Add or change a sensor probe in seconds
- Solar powered with internal and external panel options
- Radios available: Zigbee, 802.15.4, Wifi, 868MHz, 900MHz and 3G/GPRS
- Over the air programming (OTAP) of multiple nodes at once
- Special holders and brackets ready for installation in street lights and building fronts
- Graphical and intuitive programming interface

### 2.2. Sensor Probes

Sensor probes can be easily attached by just screwing them into the bottom sockets. This allows you to add new sensing capabilities to existing networks just in minutes. In the same way, sensor probes may be easily replaced in order to ensure the lowest maintenance cost of the sensor network.



Figure 1: Connecting a sensor probe to Waspmote Plug & Sense!

## 2.3. Solar Powered

Battery can be recharged using the internal or external solar panel options.

The external solar panel is mounted on a 45° holder which ensures the maximum performance of each outdoor installation.



Figure 2: Wasp mote Plug & Sense! powered by an external solar panel

For the internal option, the solar panel is embedded on the front of the enclosure, perfect for use where space is a major challenge.



Figure 3: Internal solar panel



Figure 4: Waspmote Plug & Sense! powered by an internal solar panel

## 2.4. Programming the Nodes

Waspmote Plug & Sense! can be reprogrammed in two ways:

The basic programming is done from the USB port. Just connect the USB to the specific external socket and then to the computer to upload the new firmware.



Figure 5: Programming a node

Over the Air Programming is also possible once the node has been installed. With this technique you can reprogram wirelessly one or more Wasp mote sensor nodes at the same time by using a laptop and the Wasp mote Gateway.

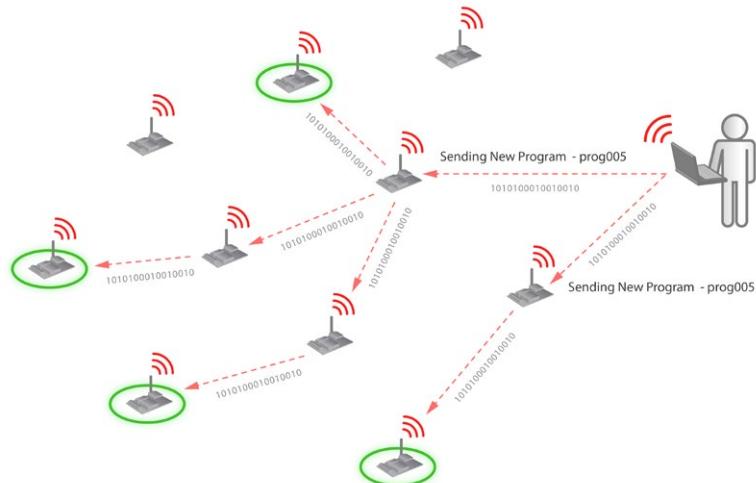


Figure 6: Typical OTAP process

## 2.5. Radio Interfaces

Model	Protocol	Frequency	txPower	Sensitivity	Range *
XBee-802.15.4-Pro	802.15.4	2.4GHz	100mW	-100dBm	7000m
XBee-ZB-Pro	ZigBee-Pro	2.4GHz	50mW	-102dBm	7000m
XBee-868	RF	868MHz	315mW	-112dBm	12km
XBee-900	RF	900MHz	50mW	-100dBm	10Km
Wifi	802.11b/g	2.4GHz	0dBm - 12dBm	-83dBm	50m-500m
GPRS	-	850MHz/900MHz/ 1800MHz/1900MHz	2W(Class4) 850MHz/900MHz, 1W(Class1) 1800MHz/1900MHz	-109dBm	
3G/GPRS	-	Tri-Band UMTS 2100/1900/900MHz Quad-Band GSM/EDGE, 850/900/1800/1900 MHz	UMTS 900/1900/2100 0,25W  GSM 850MHz/900MHz 2W  DCS1800MHz/PCS1900MHz 1W	-106dBm	

\* Line of sight, Fresnel zone clearance and 5dBi dipole antenna.

## 2.6. Program in minutes

In order to program the nodes an intuitive graphic interface has been developed. Developers just need to fill a web form in order to obtain the complete source code for the sensor nodes. This means the complete program for an specific application can be generated just in minutes. Check the Code Generator to see how easy it is at:

[http://www.libelium.com/development/plug\\_&\\_sense/sdk\\_and\\_applications/code\\_generator](http://www.libelium.com/development/plug_&_sense/sdk_and_applications/code_generator)

The screenshot shows the 'Waspmote - Plug & Sense! - Code Generator' interface. It features three main configuration sections: 'Select Model' (with a dropdown for 'Board model'), 'Sleeping Time' (with a dropdown for 'Time (seconds)'), and 'Select sensor by socket' (with dropdowns for A through F). Below these are sections for 'Additional information' (checkboxes for 'Add Accelerometer 3 Axis data' and 'Add GPS coordinates', and a field for 'Waspmote identificator (nodeID)') and 'Select Communication Module' (dropdown for 'Communication Module' set to 'USB (Debugging)'). A note indicates '\* Mandatory fields'. At the bottom right is a large 'Generate Code' button.

Figure 7: Code Generator

## 2.7. Data to the Cloud

The Sensor data gathered by the Waspmote Plug & Sense! nodes is sent to the Cloud by **Meshlium**, the Gateway router specially designed to connect Waspmote sensor networks to the Internet via Ethernet, Wifi and 3G interfaces.

Thanks to Meshlium's new feature, the Sensor Parser, now it is easier to receive any frame, parse it and store the data into a local or external Data Base.



Figure 8: Meshlium

## 2.8. Meshlium Storage Options

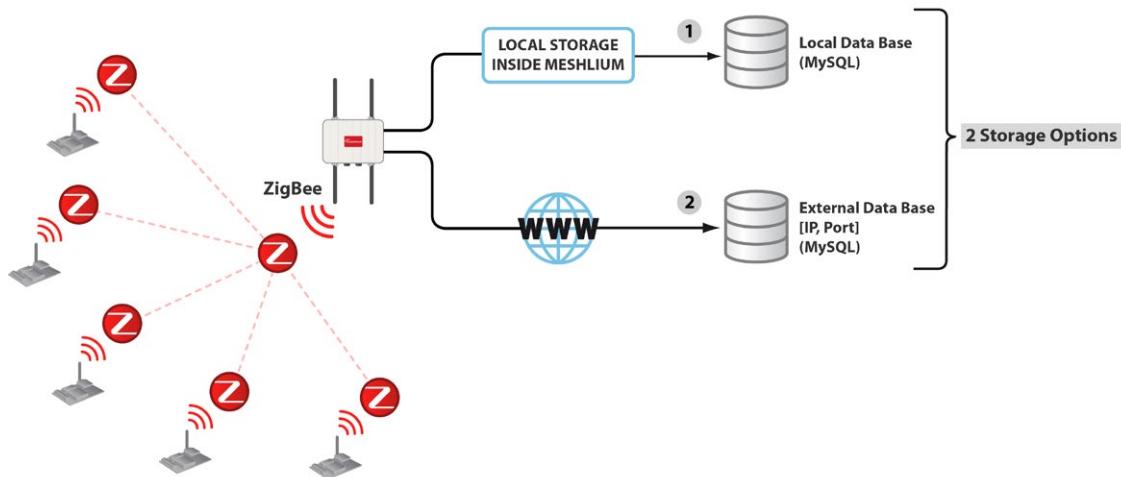


Figure 9: Meshlium Storage Options

- Local Data Base
- External Data Base

## 2.9. Meshlium Connection Options

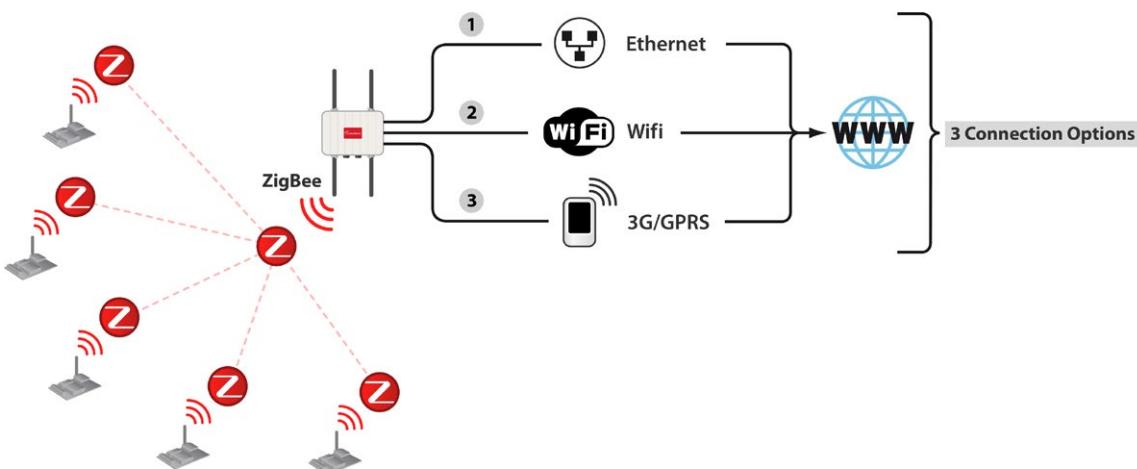


Figure 10: Meshlium Connection Options

- ZigBee → Ethernet
- ZigBee → WiFi
- ZigBee → 3G/GPRS

## 2.10. Models

There are some defined configurations of Waspmote Plug & Sense! depending on which sensors are going to be used. Waspmote Plug & Sense! configurations allow to connect up to six sensor probes at the same time.

Each model takes a different conditioning circuit to enable the sensor integration. For this reason each model allows to connect just its specific sensors.

This section describes each model configuration in detail, showing the sensors which can be used in each case and how to connect them to Waspmote. In many cases, the sensor sockets accept the connection of more than one sensor probe. See the compatibility table for each model configuration to choose the best probe combination for the application.

It is very important to remark that each socket is designed only for one specific sensor, so **they are not interchangeable**. Always be sure you connected probes in the right socket, otherwise they can be damaged.

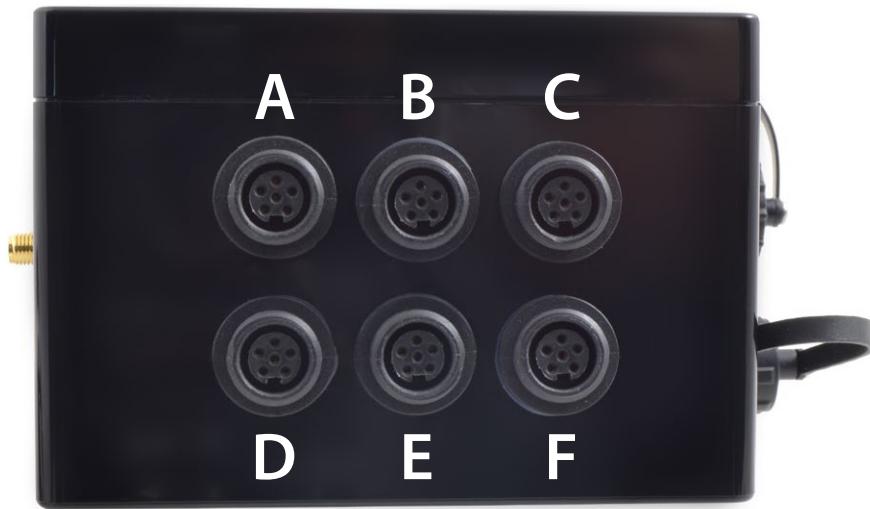


Figure 11: Identification of sensor sockets

## 2.10.1. Smart Cities

The main applications for this Waspmote Plug & Sense! model are noise maps (monitor in real time the acoustic levels in the streets of a city), air quality, waste management, structural health, smart lighting, etc. Refer to [Libelium website](#) for more information.



Figure 12: Smart Cities Waspmote Plug & Sense! model

Sensor sockets are configured as shown in the figure below.

Sensor Socket	Sensor probes allowed for each sensor socket	
	Parameter	Reference
A	Temperature	9203
	Soil temperature	86949*
	Ultrasound (distance measurement)	9246
B	Humidity	9204
	Ultrasound (distance measurement)	9246
C	Luminosity	9205
D	Noise sensor	9259
E	Dust sensor	9320
F	Linear displacement	9319

Figure 13: Sensor sockets configuration for Smart Cities model

\* Ask Libelium **Sales Department** for more information.

As we see in the figure below, thanks to the directionable probe, the ultrasound sensor probe may be placed in different positions. The sensor can be focused directly to the point we want to measure.



Figure 14: Configurations of the ultrasound sensor probe

**Note:** For more technical information about each sensor probe go to the [Development section](#) in Libelium website.

## 3. Hardware

### 3.1. General Description

The purpose of the WaspMote Smart Cities Board is to extend the monitoring functionalities of the Smart Metering Board from indoor environments to outdoor locations. Apart from the humidity, luminosity and temperature sensors, present in all the Libelium boards, other three sensors for specific applications have been included: three sensors destined to monitor cracks in buildings and structures, a linear displacement sensor (SLS095) for crack width, a single strand strain gage for crack detection and a multiple strand strain gage for crack propagation. Also a dust and PM-10 particles sensor (GP2Y1010AU0F) has been introduced, used to measure the concentration of particles in suspension in the environment in air quality control applications, and finally the WM-61A microphone, adapted to measure the environmental noise in the A decibels scale.

### 3.2. Specifications

**Weight:** 20gr

**Dimensions:** 73.5 x 51 x 1.3 mm

**Temperature Range:** [-20°C, 65°C]

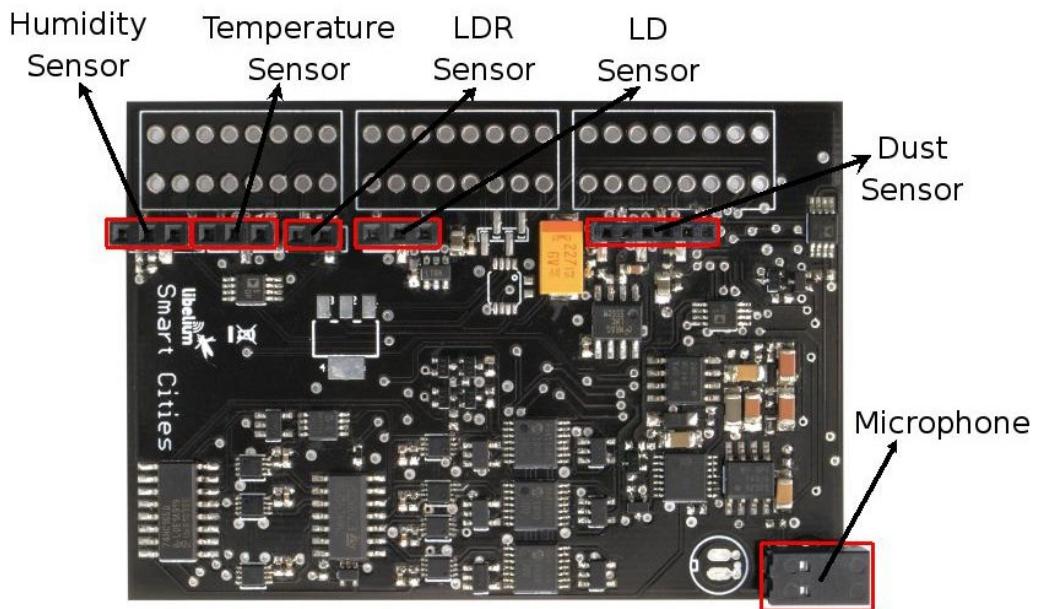


Figure 15: Upper side

### 3.3. Electrical Characteristics

**Board Power Voltages:** 3.3V and 5V

**Sensor Power Voltages:** 3.3V and 5V

**Maximum admitted current (continuous):** 200mA

**Maximum admitted current (peak):** 400mA

## 4. Sensors

### 4.1. Particle Sensor (PM-10) – Dust Sensor (GP2Y1010AU0F)

#### 4.1.1. Specifications

**Supply voltage:** -0.3V ~ 7V

**Sensitivity:** Typical: 0.5V/(0.1mg/m<sup>3</sup>), Minimum: 0.35V/(0.1mg/m<sup>3</sup>), Maximum: 0.65V/(0.1mg/m<sup>3</sup>)

**Output voltage at no dust:** Typical: 0.9V, Minimum: 0V, Maximum: 1.5V

**Output voltage range:** 3.4V

**Operation temperature:** -10°C ~ +65°C

**Current consumption:** Typical: 11mA, Maximum: 20mA



Figure 16: GP2Y1010AU0F Dust Sensor

**LED Pulse Cycle:** 10±1ms

**LED Pulse width:** 0.32±0.02ms

**LED Operating supply voltage:** 5±0.5V

#### 4.1.2. Measurement Process

The GPY21010AU0F is an optical sensor whose principle of operation is based on the detection of the infrared light emitted by an ILED diode, reflected by the dust particles and captured by means of a phototransistor. The ILED diode needs to be supplied with a signal of pulses of 0.32ms width and a period of 10ms, generated automatically by the hardware of the board when the sensor is turned on, being the output a signal of pulses of the same time characteristics whose amplitude is proportional to the environmental dust density (see the graph in figure 17). To read this signal has been added a demodulation circuit that extracts the envelope of the train of pulses at whose output results an analog voltage in a range between 0V and 3V approximately that can be read at one of the analog inputs of the mote (ANALOG1). The supply voltage is controlled through a solid state switch activated with the signal DIGITAL2.

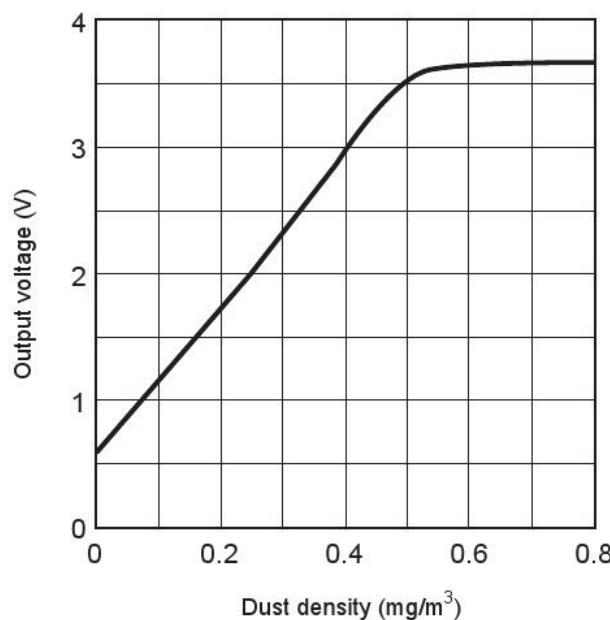


Figure 17: Graph of the output voltage vs dust density extracted from the Sharp's sensor data sheet



Figure 18: Example of application for the particle sensor

Below a sample code for reading the output of the sensor and converting the voltage measured into a dust density value using the libraries of the board is shown:

```
{
SensorCities.ON();
SensorCities.setSensorMode(SENS_CITIES_DUST, SENS_ON);
delay(2000);
float dust_value;
dust_value = SensorCities.readValue(SENS_CITIES_DUST);
}
```

You can find a complete example code for reading the dust sensor in the following link:

<http://www.libelium.com/development/wasp mote/examples/sc-4-dust-sensor-reading>

### 4.1.3. Socket

In figure 19 we can see an image of the socket for the sensor (6 ways, 2mm pitch) with the pin correspondence between them highlighted. In section "Sockets for casing" more information about the corresponding pinout at the sockets for casing applications can be found.

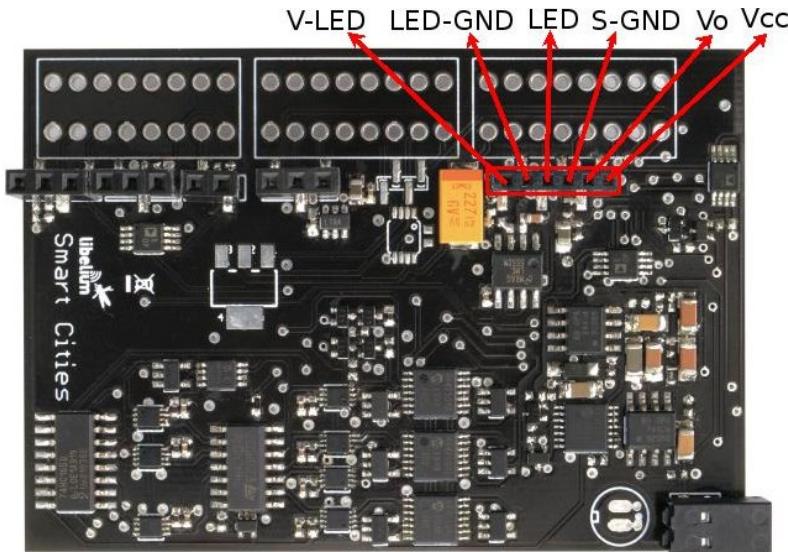


Figure 19: Image of the socket for the GP2Y1010AU0F sensor

In the figure below we can see the sensor pinout corresponding with the connector shown above. The recommended connector for this sensor is the ZHR-6(P) along with six crimp terminals SZH-002T-P0.5, both of them by JST.

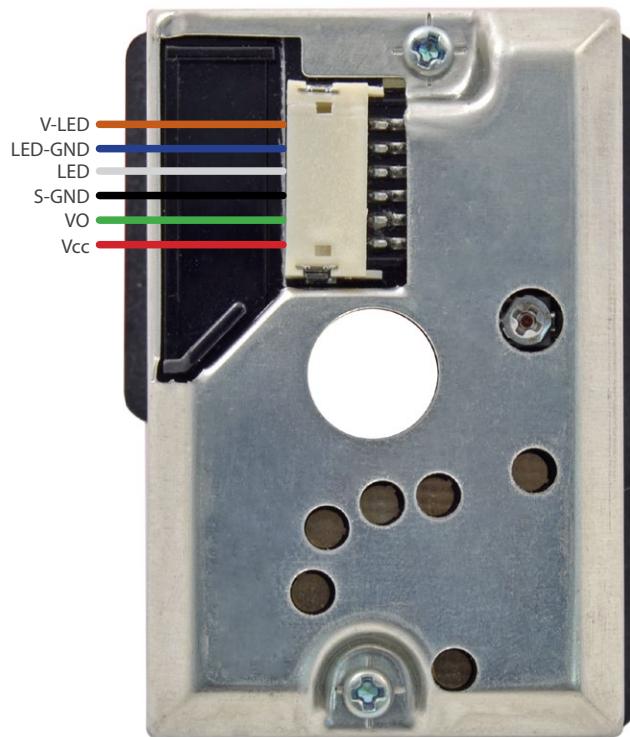


Figure 20: Image of the GP2Y1010AU0F sensor with the pinout indicated

**Note:** this sensor is sold ready to use only in the Plug & Sense! line, the user who wills to connect it directly to the Smart Cities Board will have to add the connector necessary to wire it and plug it into its socket.

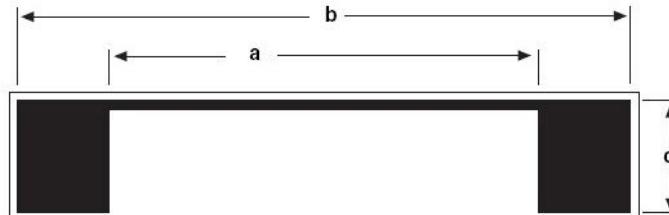
## 4.2. Crack detection sensors (Vishay)

### 4.2.1. Specifications

**Operating temperature:** -195°C~120°C



Figure 21: Image of the crack detection sensor



GAGE DESIGNATION	DIMENSIONS				
	a	b	c	Length	Width
CD-02-10A CD-23-10A	0.40	0.56	0.10	0.60	0.13
	10.2	14.2	2.5	15.2	3.2
CD-02-15A CD-23-15A	0.60	0.76	0.10	0.80	0.13
	15.2	19.3	2.5	20.3	3.2
CD-02-20A CD-23-20A	0.80	0.96	0.10	1.00	0.13
	20.3	24.4	2.5	25.4	3.2
CD-02-25A CD-23-25A	1.00	1.16	0.10	1.20	0.13
	25.4	29.5	2.5	30.5	3.2
CD-02-50A CD-23-50A	2.00	2.16	0.10	2.22	0.13
	50.8	54.9	2.5	56.4	3.2

Figure 22: Dimensions of the crack detection sensor extracted from the datasheet of the Vishay sensor

## 4.2.2. Measurement Process

The crack propagation sensor consists of a small conductive strand with a very low resistance value embedded in a fiber-glass film, when the sensor remains intact it sets a logic 'one' in a digital input of the WaspMote. In presence of a crack, the sensor shall break, turning to a logic 'zero' in the input pin of the microcontroller (ANALOG5). The sensor must be fixed to the surface using a special adhesive. being recommended the use of a protective coating in long term installations.

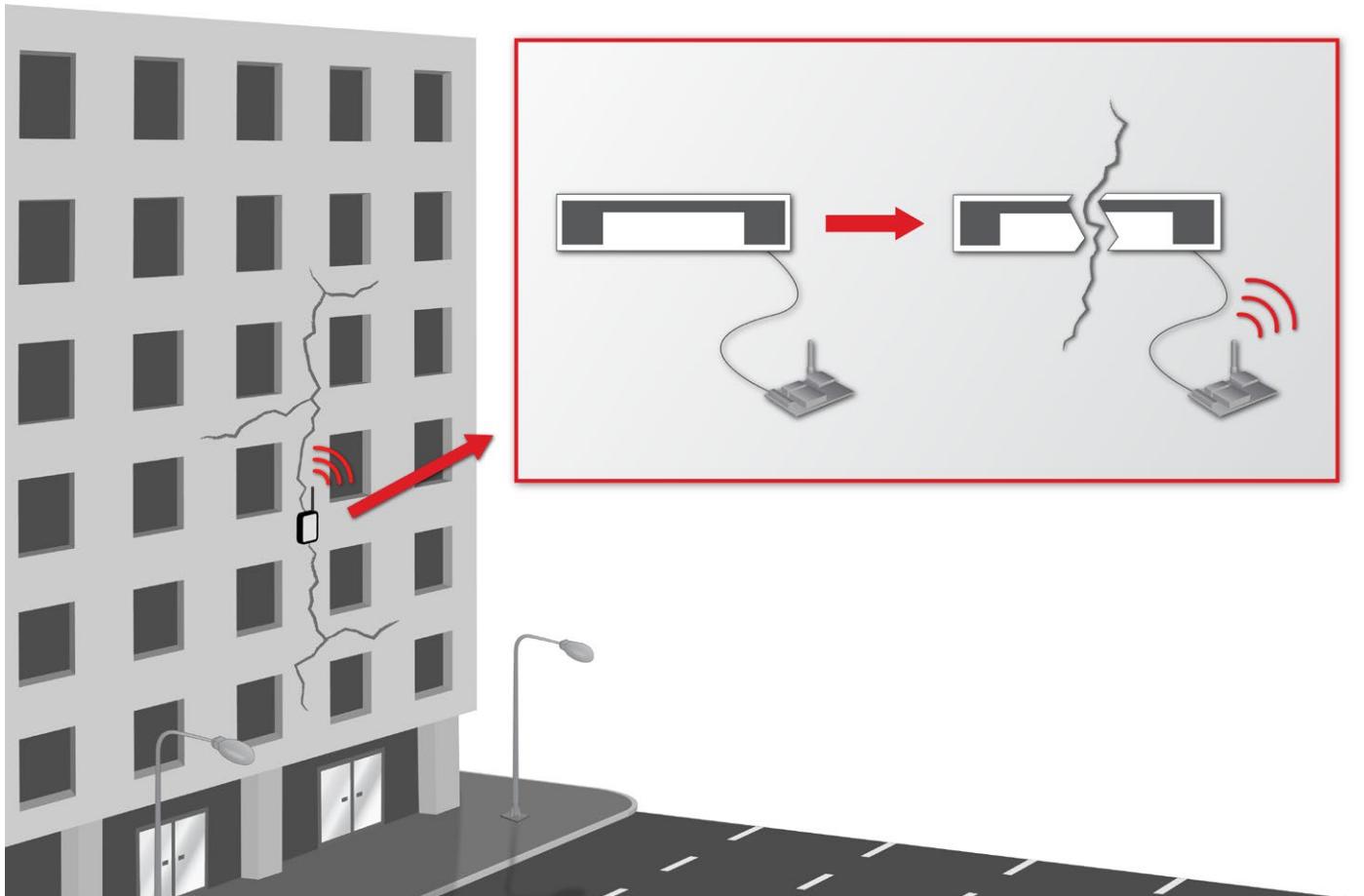


Figure 23: Example of application for the crack detection sensor

Below a code to measure the sensor is shown:

```
{
  SensorCities.setBoardMode(SENS_ON);
  SensorCities.setSensorMode(SENS_CITIES_CD, SENS_ON);
  delay(100);
  float crack_value;
  crack_value = SensorCities.readValue(SENS_CITIES_CD);
}
```

You can find a complete example code for reading the crack detection sensor in the following link:

<http://www.libelium.com/development/wasp mote/examples/sc-7-crack-detection-sensor-reading>

### 4.2.3. Socket

This sensor shares the socket with the luminosity sensor (LDR), upon which may be connected independently of the pin position, since no polarity is required. In section “Sockets for casing” more information about the corresponding pinout at the sockets for casing applications can be found.

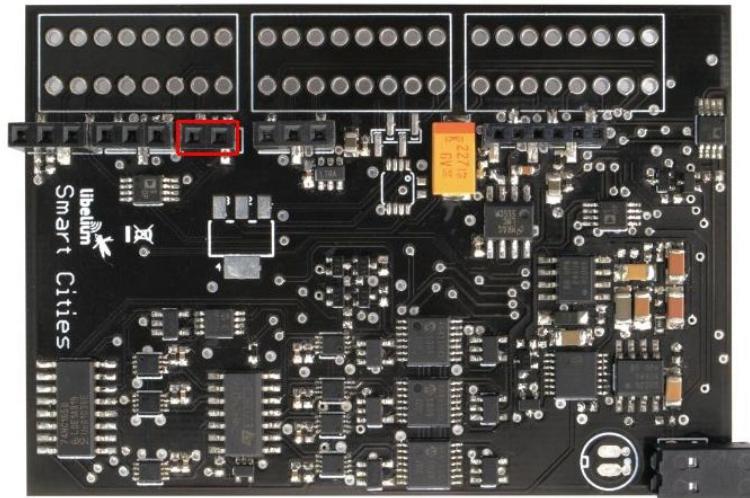


Figure 24: Image of the socket for the crack detection sensor

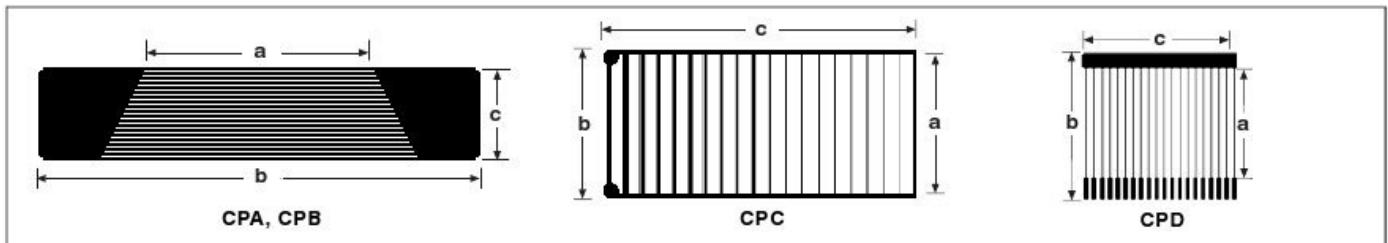
## 4.3. Crack propagation sensors (Vishay)

### 4.3.1. Specifications

**Operating temperature:** -195°C~120°C



Figure 25: Image of the crack propagation sensor



GAGE PATTERN AND DESIGNATION	 RoHS COMPLIANT	NOMINAL RESISTANCE IN OHMS	DIMENSIONS				
			a	b	c	MATRIX	
						Length	Width
TK-09-CPB02-005/DP		5	0.25	0.50	0.10	0.56	0.16
			6.4	12.7	2.5	14.2	4.1
Ten Grid Lines — 0.010 in [0.25 mm] between centerlines.							
TK-09-CPA01-005/DP		5	0.50	1.00	0.20	1.08	0.28
			12.7	25.4	5.1	27.4	7.1
Twenty Grid Lines — 0.010 in [0.25 mm] between centerlines.							
TK-09-CPA02-005/DP		5	1.00	2.00	0.40	2.08	0.48
			25.4	50.8	10.2	52.8	12.2
Twenty Grid Lines — 0.020 in [0.51 mm] between centerlines.							
TK-09-CPC03-003/DP		3	0.70	0.75	1.57	0.80	1.62
			17.8	19.1	39.9	20.3	41.1
Twenty Grid Lines — 0.080 in [2.03 mm] between centerlines.							
TK-09-CPD01-NRA/DP		110	0.75	1.00	1.00	1.11	1.11
			19.1	25.4	25.4	28.1	28.1
Twenty Grid Lines — 0.050 in [1.27 mm] between centerlines.							

Figure 26: Dimensions of the crack propagation sensor extracted from the datasheet of the Vishay sensor

### 4.3.2. Measurement Process

The crack propagation sensor is based in the same principle of operation that the crack detection sensor, save that it is composed of several resistive strands in parallel whose breakage causes a variation in the total resistance of the sensor following the pattern shown in figure 28. That resistance can be measured through a voltage divider, so a voltage proportional to the number of broken strands is obtained at the analog input ANALOG5.

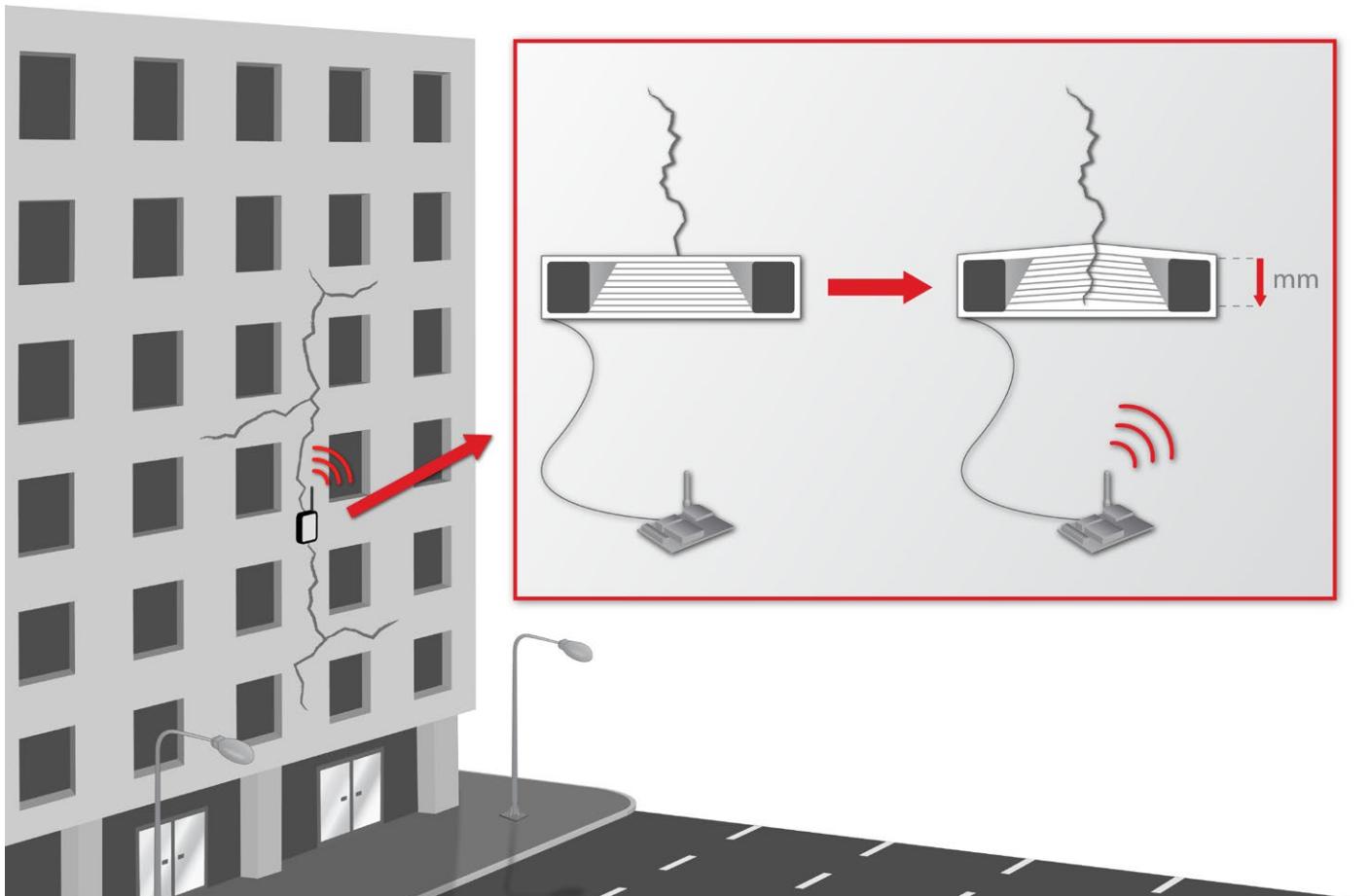


Figure 27: Example of application for the crack propagation sensor

Below a code to measure the sensor is shown:

```
{
  SensorCities.setBoardMode(SENS_ON);
  SensorCities.setSensorMode(SENS_CITIES_CP, SENS_ON);
  delay(100);
  float crack_value;
  crack_value = SensorCities.readValue(SENS_CITIES_CP);
}
```

You can find a complete example code for reading the crack detection sensor in the following link:

<http://www.libelium.com/development/wasp mote/examples/sc-6-crack-propagation-sensor-reading>

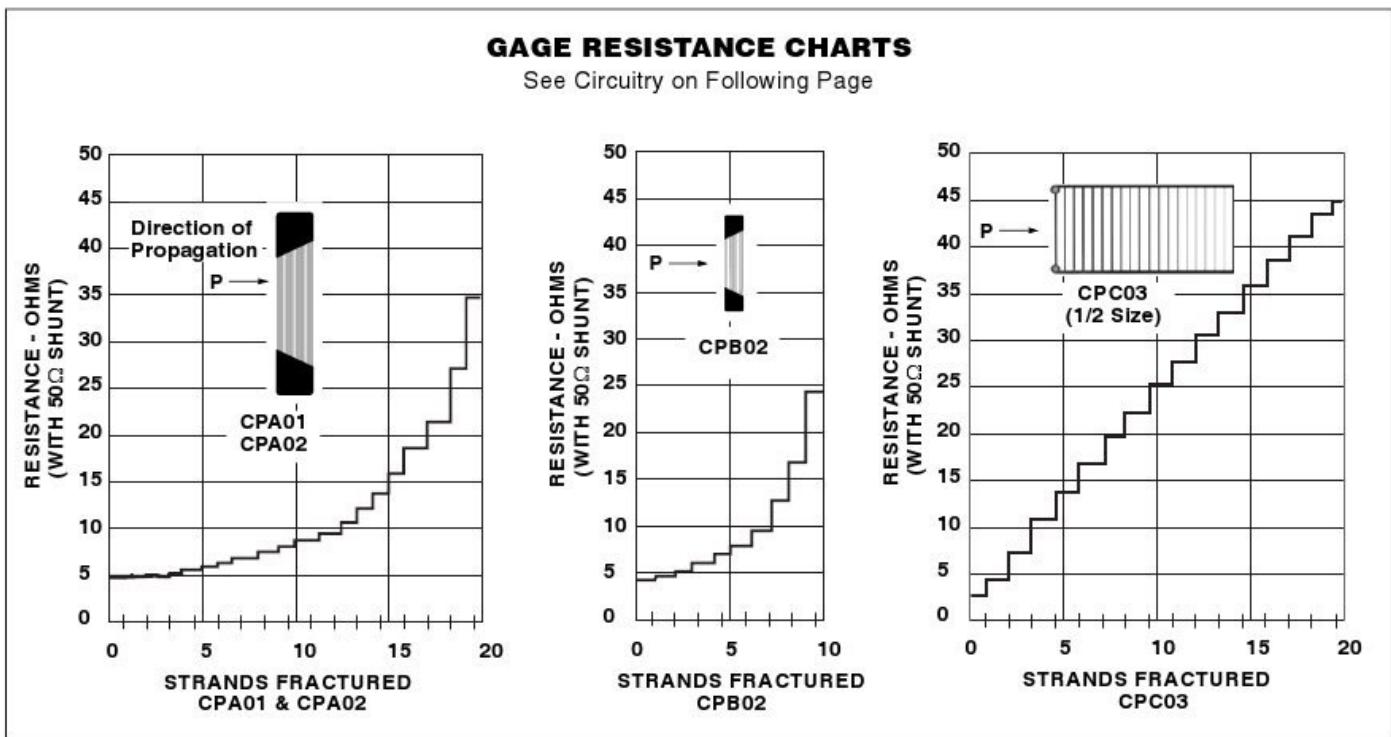


Figure 28: Variation of the crack propagation sensor resistance extracted from the data sheet of the Vishay sensor

### 4.3.3. Socket

The crack propagation sensor must be placed in the same socket that the LDR and the crack detection sensor. Like those two sensors, the pin correspondence is not relevant, but in this case a change in the load resistor is necessary for a proper operation (please contact the Libelium [Sales department](#) when acquiring the Smart Cities board along with this sensor).

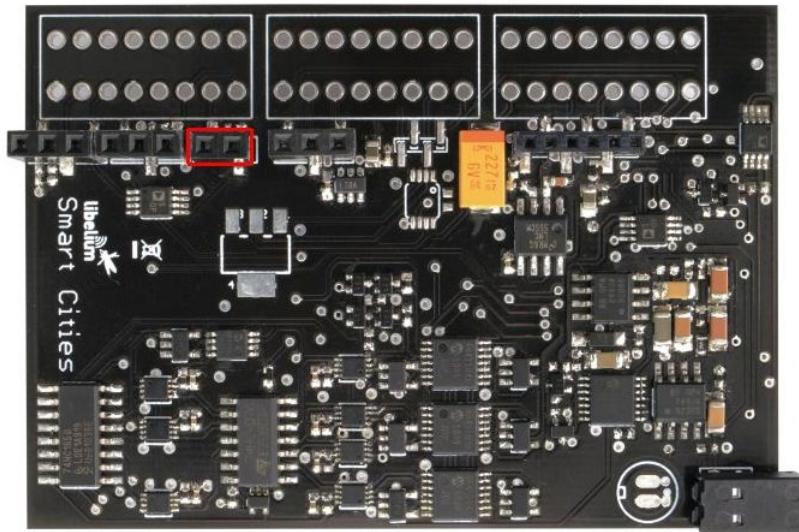


Figure 29: Image of the socket for the crack propagation sensor

## 4.4. Linear Displacement Sensor - Crack measurement (SLS095)

### 4.4.1. Specifications

**Electrical stroke:** 10mm  
**Sensor resistance:** 400Ω  
**Linearity:** ±0.5%  
**Resolution:** 10µm (imposed by the analog-to-digital conversion)  
**Supply Voltage:** +8.9V  
**Power dissipation (20°C):** 0.2W  
**Temperature Operation:** -30°C ~ 100°C



Figure 30: SLS095 displacement sensor

### 4.4.2. Measurement Process

The SLS095 linear displacement sensor by Penny+Giles is a potentiometer whose wiper moves along with an axis guided by the sensor's body. Fixing both ends of the potentiometer at the sides of the crack we can measure its width by reading the voltage at the wiper. For this, the sensor has been configured as a voltage divider, with one of the ends sourced from a 3V supply, the other end grounded and the wiper connected to the input ANALOG7 of the analog-to-digital converter of the WaspMote, which leads to a resolution of 11µm approximately. The supply voltage comes from a solid state switch controlled by the pin DIGITAL1.

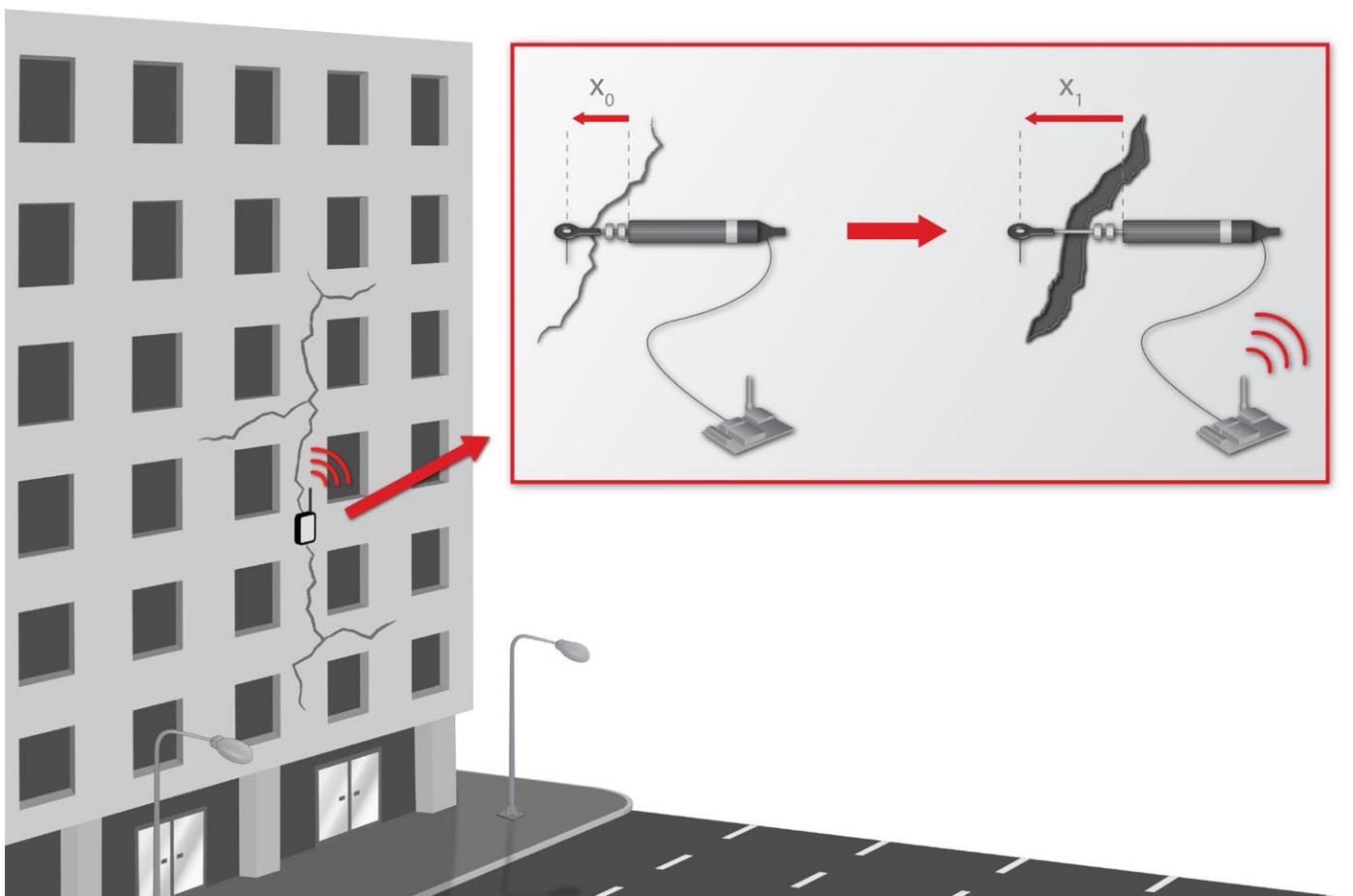


Figure 31: Example of application for the linear displacement sensor

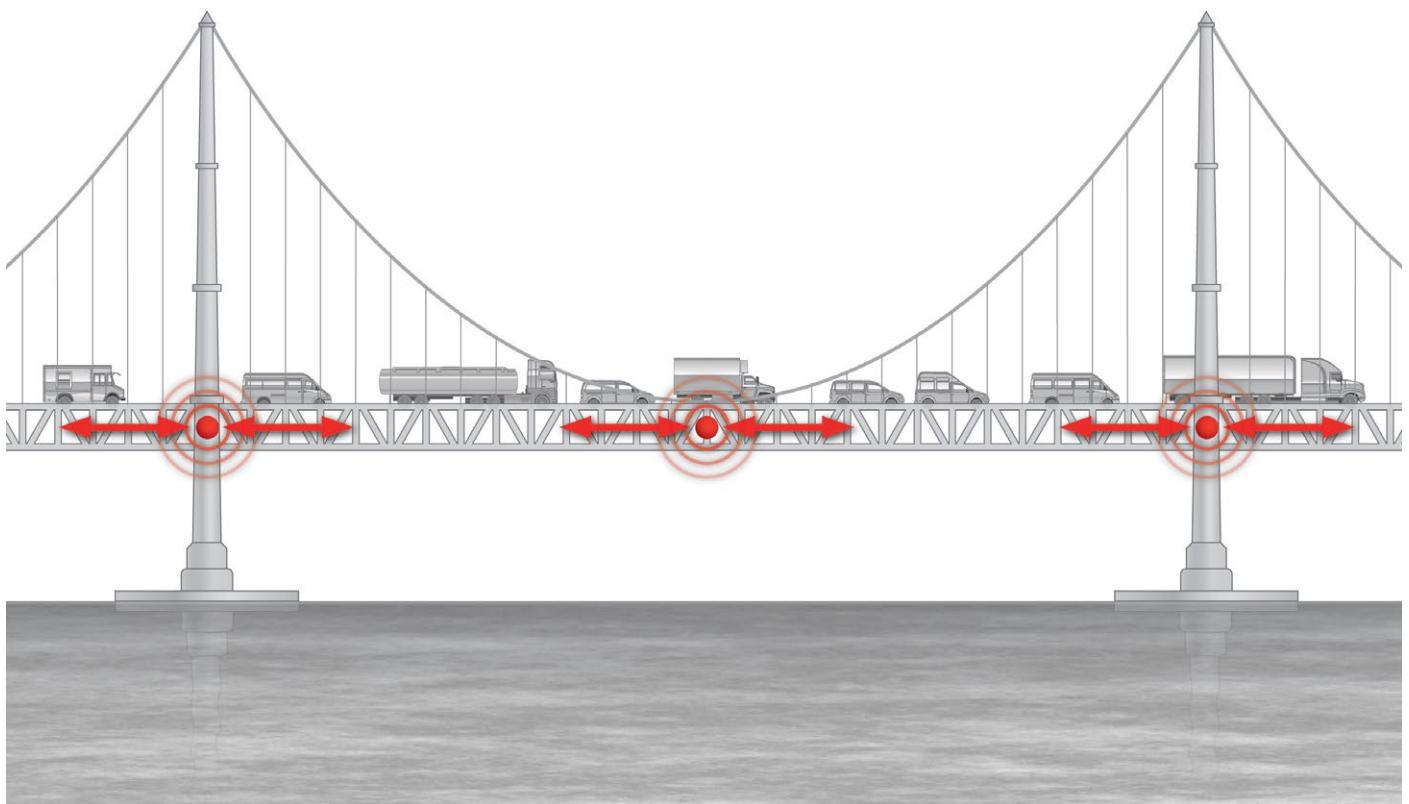


Figure 32: Example of measurement of expansion and contraction of a bridge for the linear displacement sensor

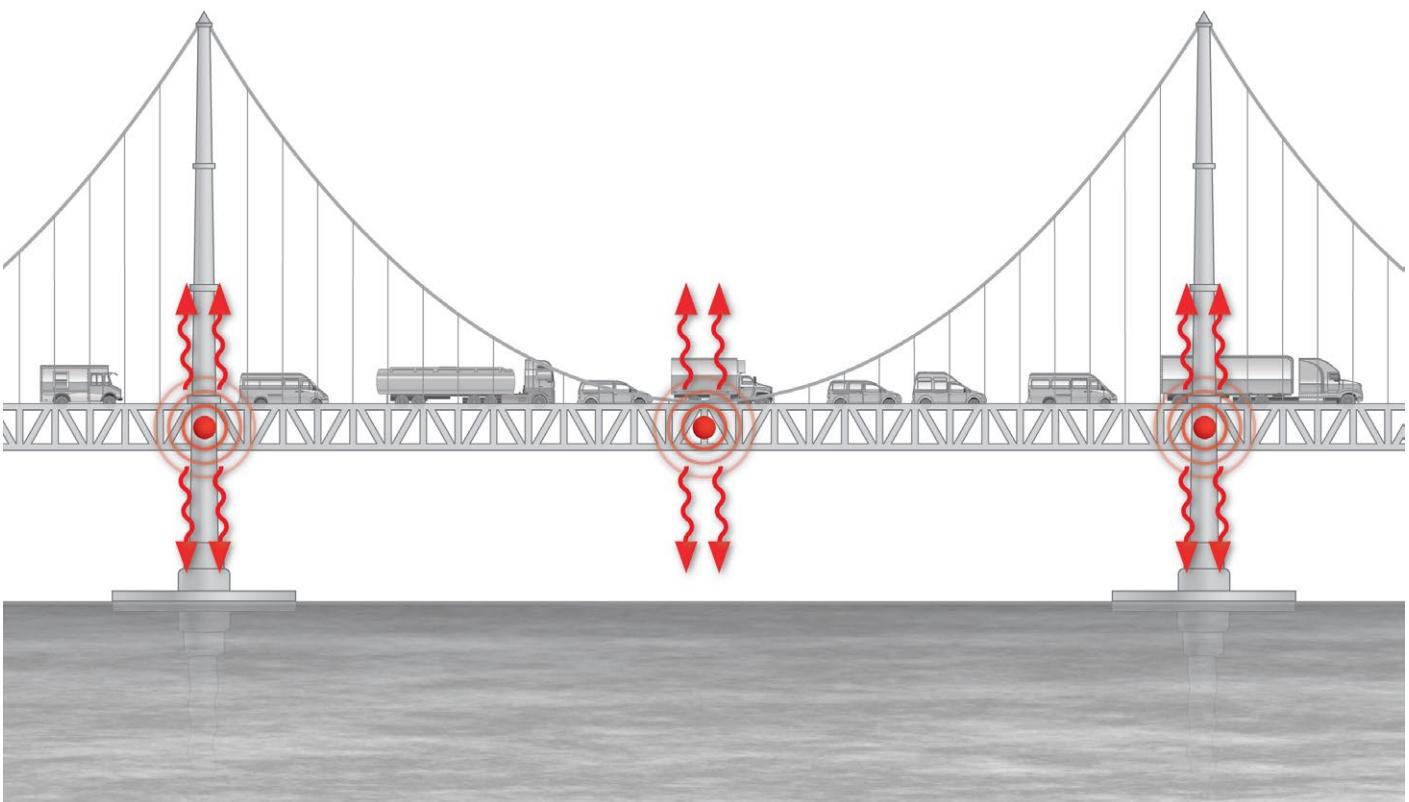


Figure 33: Example of measurement of vibration in a bridge for the linear displacement sensor. The vibration measurement is complemented by the accelerometer integrated in WaspMote

Below a sample code for reading the output of the sensor and converting the voltage measured into micrometers using the libraries of the board is shown:

```
{
SensorCities.setBoardMode(SENS_ON);
SensorCities.setSensorMode(SENS_CITIES_LD, SENS_ON);
delay(100);
float ld_value;
crack_sensor_value = SensorCities.readValue(SENS_CITIES_LD);
}
```

You can find a complete example code for reading the linear displacement sensor in the following link:

<http://www.libelium.com/development/wasp mote/examples/sc-5-crack-sensor-reading>

#### 4.4.3. Socket

We can see an image of the socket for the sensor in figure 34 (3 ways, 2.54mm pitch) where the correspondence with the pins of the sensor is indicated. The red and the black wires of the SLS095 corresponds with the two ends of the potentiometer (interchangeable with ground or power supply connections), while the yellow wire corresponds with its wiper. In section "Sockets for casing" more information about the corresponding pinout at the sockets for casing applications can be found.

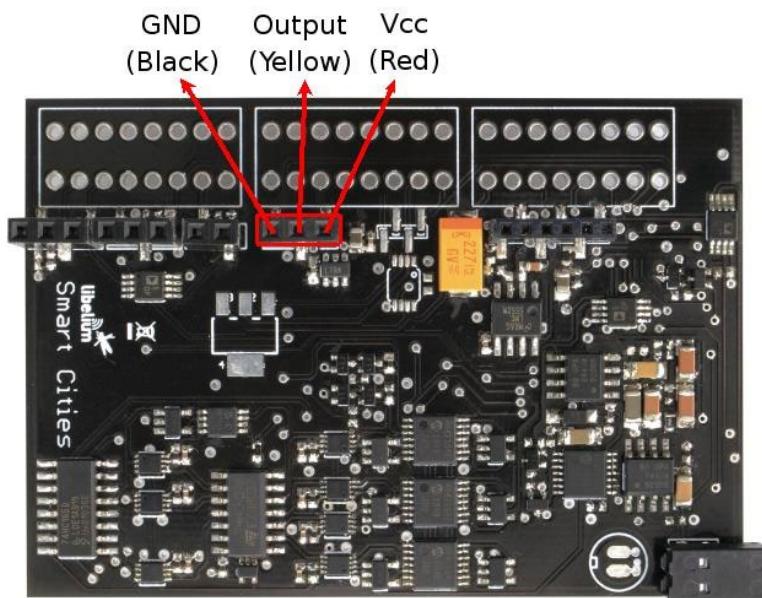


Figure 34: Image of the socket for the SLS095 sensor

## 4.5. Noise Sensor (Microphone WM-61A)

### 4.5.1. Specifications

**Microphone specifications:**

**Sensitivity:**  $-35 \pm 4 \text{ dB}$

**Impedance:**  $< 2.2 \text{ k}\Omega$

**Directivity:** Omnidirectional

**Frequency:**  $20 \text{ Hz} \sim 20 \text{ kHz}$

**Supply voltage:**  $+2 \text{ V}$  (Standard),  $+10 \text{ V}$  (Maximum)

**Maximum current consumption:**  $0.5 \text{ mA}$

**Sensitivity reduction:**  $-3 \text{ dB}$  at  $1.5 \text{ V}$

**Maximum sound pressure level:**  $114.5 \pm 10 \text{ dB SPL}$  approximately

**S/N ratio:**  $62 \text{ dB}$

**Noise Level:**  $26 \pm 1 \text{ dB SPL}$

**Stage Measurement range:**  $50 \text{ dB SPL} \sim 100 \text{ dB SPL}$



Figure 35: Image of the WM-61A microphone by Panasonic

### 4.5.2. Measurement Process

The WM-61A, introduced in the Smart Cities board to monitor the environmental noise, is an omnidirectional microphone with an almost flat response in the whole frequency range of human hearing, between  $20 \text{ Hz}$  and  $20 \text{ kHz}$ . From the characteristics shown in the specifications we can infer a noise level of  $26 \text{ dB} \pm 1 \text{ dB SPL}$  approximately and a dynamic range of  $79.5 \text{ dB}$  approximately. A circuit to filter the signal to adapt it to the A decibel scale and output a continuous voltage readable from the mote's processor (at the analog input pin ANALOG6) has been introduced. When sold along with a microphone, the Smart Cities board is supplied calibrated by Libelium to return an output in the range between  $50 \text{ dB SPL}$  and  $100 \text{ dB SPL}$  with an accuracy of  $\pm 2.5 \text{ dB SPL}$ . The calibration data associated to the microphone reading is stored in the microcontroller's EEPROM, between addresses 164 and 185. **Be very careful not to overwrite this memory positions** or it could lead to an irreparable error when reading this sensor.

The A weighting for the audio measurements is a compensation curve that is used to fit the sound pressure measurement to the ear response in function of the frequency, and is the most common standard for noise measurement. In figure 36 we can see a table of noise pressure generated by different sources in  $\text{dB SPL}$ .

The supply voltage of the microphone and its electronics may be turned on or off with a solid state switch controlled from the mote's processor by the DIGITAL6 output pin.

Sound	dBSPLA
Audition threshold	0
Quiet Room	30
Normal conversation	60~70
Heavy traffic (hearing loss under continued exposure)	90
Pain threshold	130
Jet engine (permanent damage)	140

Figure 36: Noise in  $\text{dB SPL}$  produced by different sources

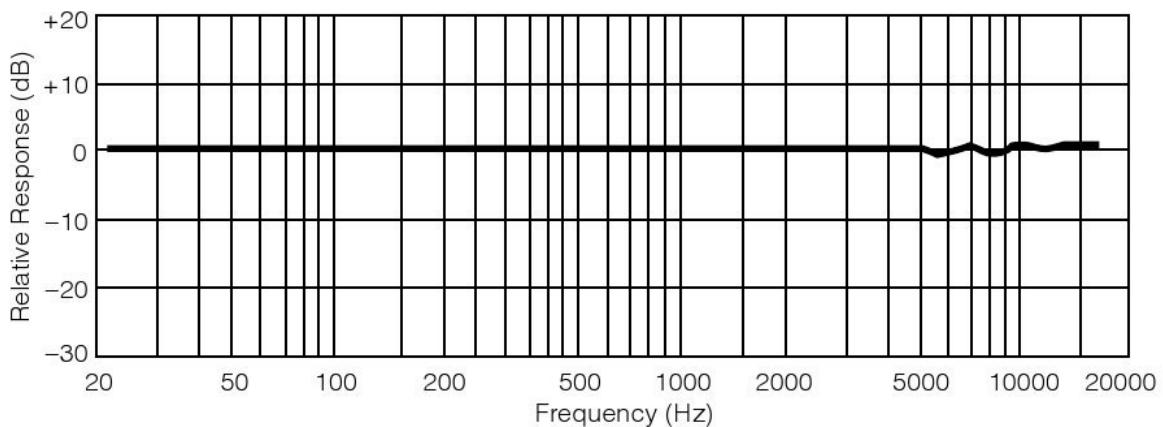


Figure 37: Graph of the frequency response of the WM-61A extracted from the sensor's data sheet from Panasonic

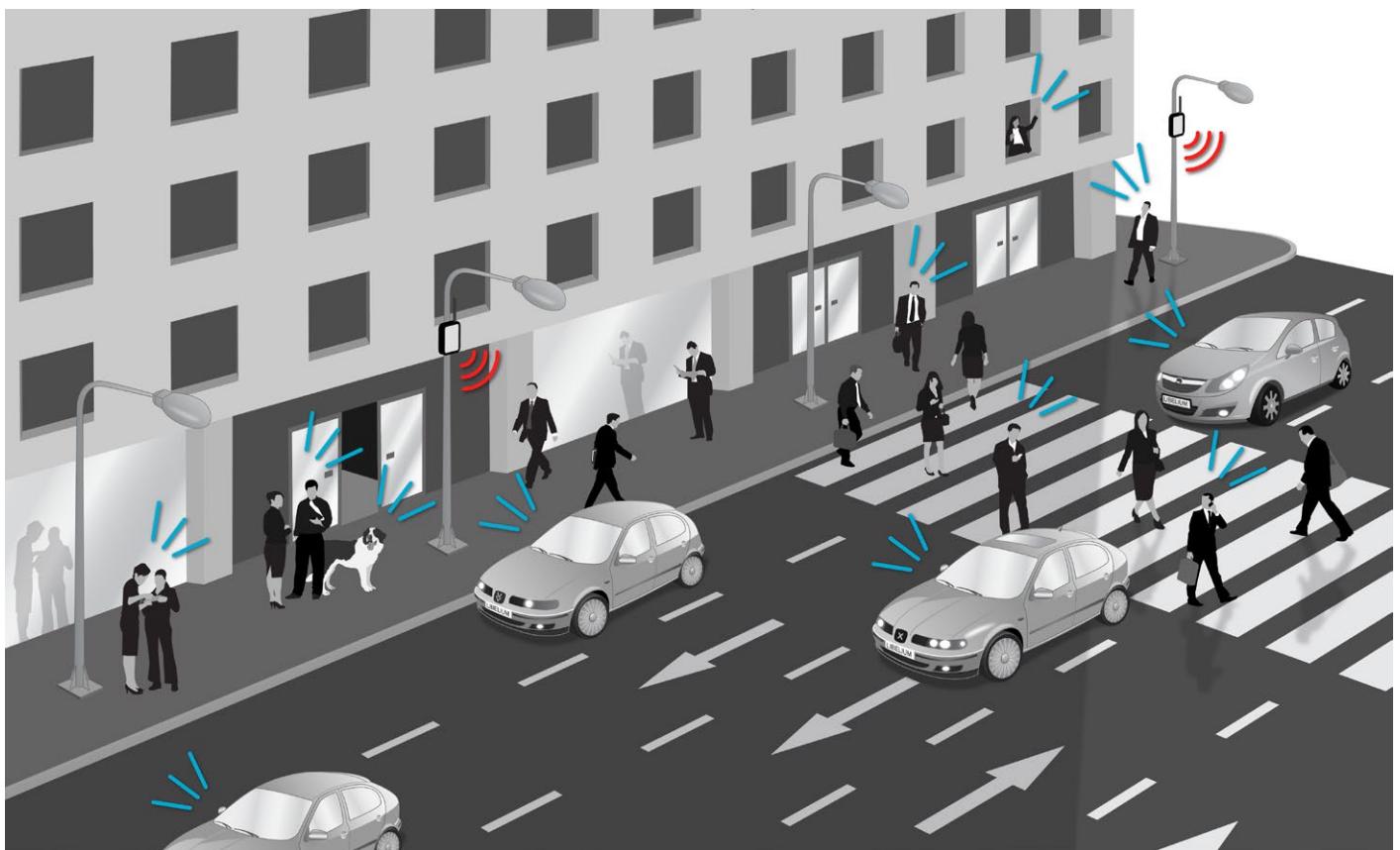


Figure 38: Example of application for the noise sensor

Below a code to measure the sensor is shown:

```
{
  SensorCities.setBoardMode(SENS_ON);
  SensorCities.setSensorMode(SENS_CITIES_AUDIO, SENS_ON);
  delay(100);
  float audio_value;
  audio_value = SensorCities.readValue(SENS_CITIES_AUDIO);
}
```

You can find a complete example code for reading the microphone in the following link:

<http://www.libelium.com/development/wasp mote/examples/sc-9-audio-sensor-reading>

### 4.5.3. Socket

The WM-61A microphone may be connected to the board in three different ways. Directly welded to the board, in the position highlighted in blue in figure 39, through the terminal block highlighted in the image in red, or through the connectors for casing, for which more information is given in section "Sockets for casing".

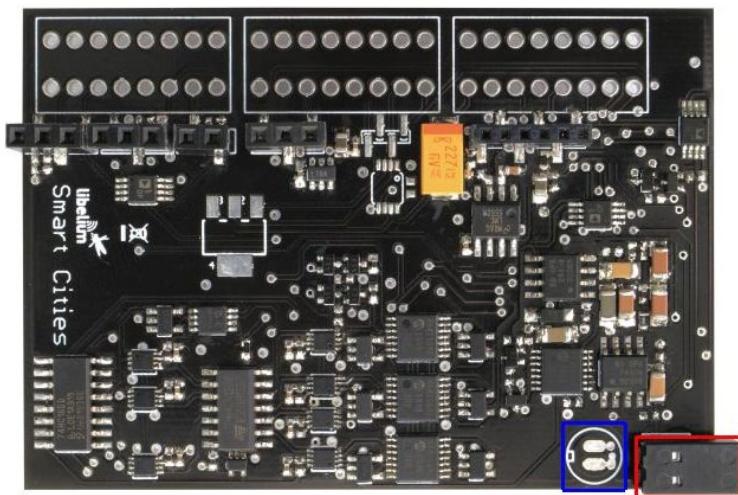


Figure 39: Image of the sockets for the WM-61A microphone

## 4.6. Ultrasonic Sensor (MaxSonar® from MaxBotix™)

### 4.6.1. Specifications

#### XL-MaxSonar®-WRA1™:

**Operation frequency:** 42kHz

**Maximum detection distance:** 765cm

**Maximum detection distance (analog output):** 600cm (powered at 3.3V) - 700cm  
(powered at 5V)

**Sensitivity (analog output):** 3.2mV/cm (powered at 3.3V) – 4.9mV/cm  
(powered at 5V)

**Power supply:** 3.3 ~ 5V

**Consumption (average):** 2.1mA (powered at 3.3V) – 3.2mA (powered at 5V)

**Consumption (peak):** 50mA (powered at 3.3V) – 100mA (powered at 5V)

**Usage:** Indoors and outdoors (IP-67)



Figure 40: Ultrasonic XL-MaxSonar®-WRA1 from MaxBotix™ sensor

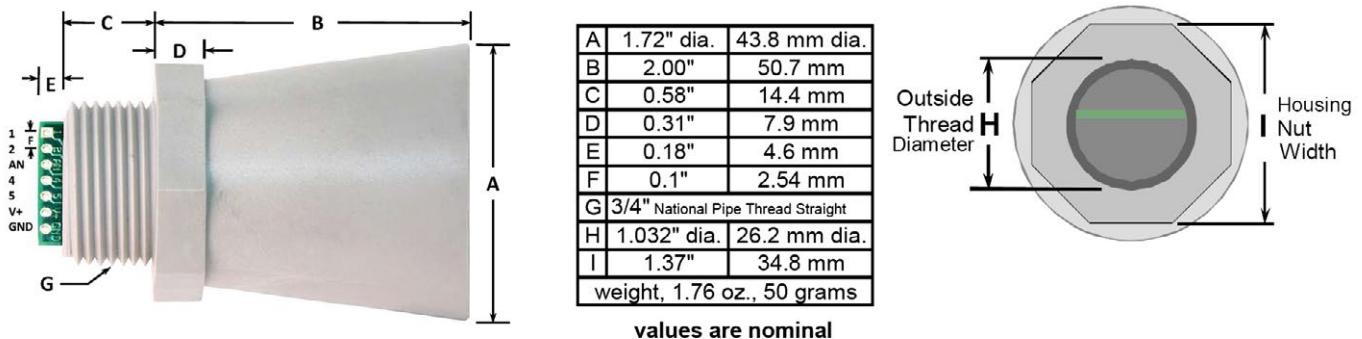


Figure 41: Ultrasonic XL-MaxSonar®-WRA1 sensor dimensions

In the figure below we can see a diagram of the detection range of the sensor developed using different detection patterns (a 0.63cm diameter dowel for diagram A, a 2.54cm diameter dowel for diagram B, a 8.25cm diameter rod for diagram C and a 28cm wide board for diagram D):

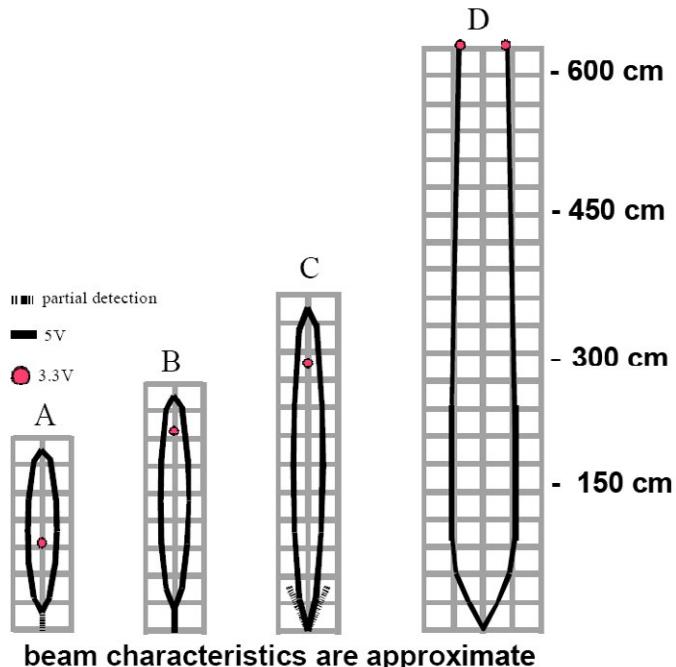


Figure 42: Diagram of the sensor beam extracted from the data sheet of the XL-MaxSonar®-WRA1™ sensor from MaxBotix

#### LV-MaxSonar®-EZ0™:

- **Operation frequency:** 42kHz
- **Maximum detection distance:** 645cm
- **Sensitivity (analog output):** 2.5mV/cm (powered at 3.3V) – 3.8mV/cm (powered at 5V)
- **Power supply:** 3.3 ~ 5V
- **Consumption (average):** 2mA (powered at 3.3V) – 3mA (powered at 5V)
- **Usage:** Indoors



Figure 43: Ultrasonic LV-MaxSonar®-EZ0 from MaxBotix™ sensor

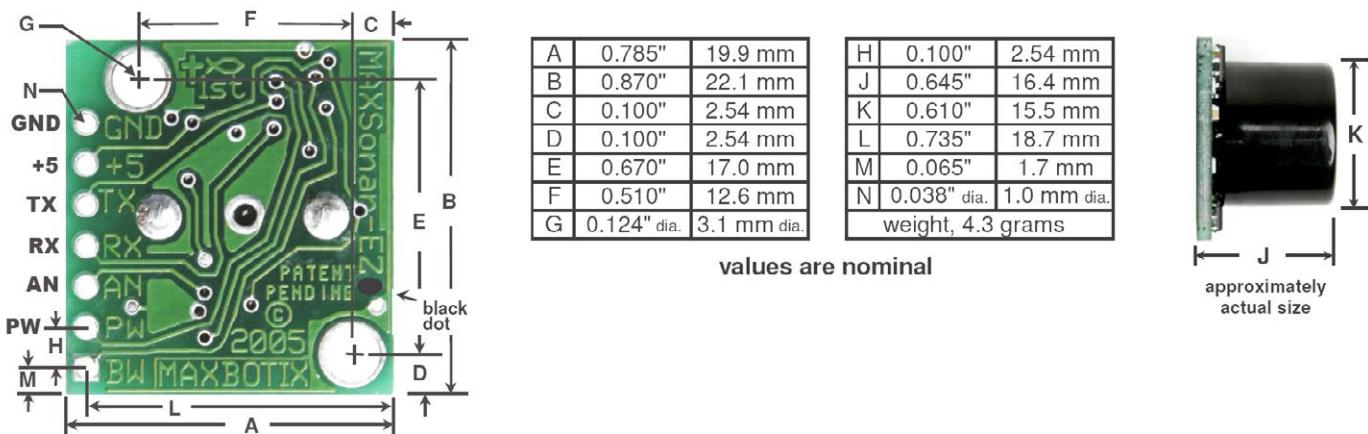


Figure 44: Ultrasonic LV-MaxSonar®-EZ0 sensor dimensions

In the figure below we can see a diagram of the detection range of the sensor developed using different detection patterns (a 0.63cm diameter dowel for diagram A, a 2.54cm diameter dowel for diagram B, a 8.25cm diameter rod for diagram C and a 28cm wide board for diagram D):

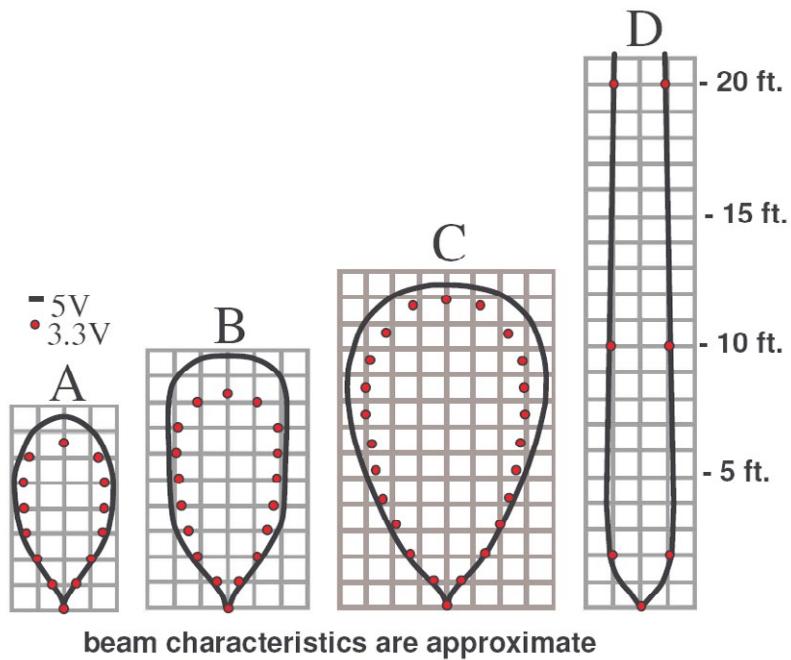


Figure 45: Diagram of the sensor beam extracted from the data sheet of the LV-MaxSonar®-EZ0™ sensor from MaxBotix

#### 4.6.2. Measurement Process

The MaxSonar® sensors from MaxBotix outputs an analog voltage proportional to the distance to the object detected. That voltage may be read through the analog input ANALOG4, while pin DIGITAL2 can be used to activate or deactivate the power supply of the sensor.

In figure 46 we can see a drawing of an example application for the ultrasonic sensors.

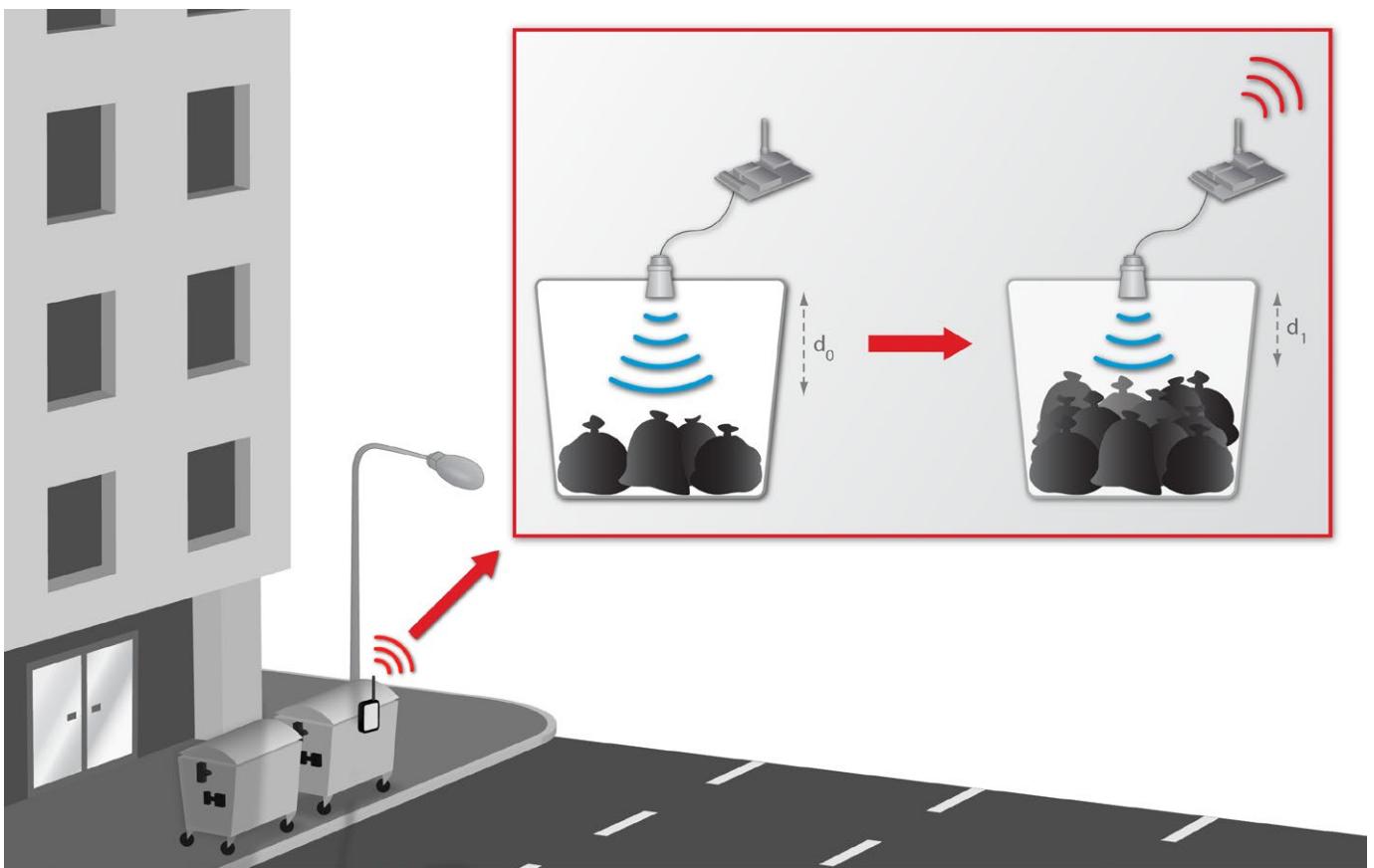


Figure 46: Example of application for the MaxSonar® sensors

The garbage levels in bins can be controlled in order to ensure an efficient waste collection system.

Below a sample code to measure one of the ultrasound sensors (the XL-MaxSonar®-WRA1) is shown:

```
{
  SensorCities.setBoardMode(SENS_ON);
  SensorCities.setSensorMode(SENS_CITIES_ULTRASOUND_3V3, SENS_ON);
  delay(2000);
  float distance_value;
  distance_value = SensorCities.readValue(SENS_CITIES_ULTRASOUND_3V3, SENS_US_WRA1);
}
```

You can find a complete example code for reading the humidity in the following link:

<http://www.libelium.com/development/wasp mote/examples/sc-8-ultrasound-sensor-reading>

### 4.6.3. Socket

This sensors share the sockets with the MCP9700A temperature sensor or the humidity 808H5V5 sensor. The pin correspondence, highlighted in figure 47, is the same for both. In section "Sockets for casing" more information about the corresponding pinout at the sockets for casing applications can be found.

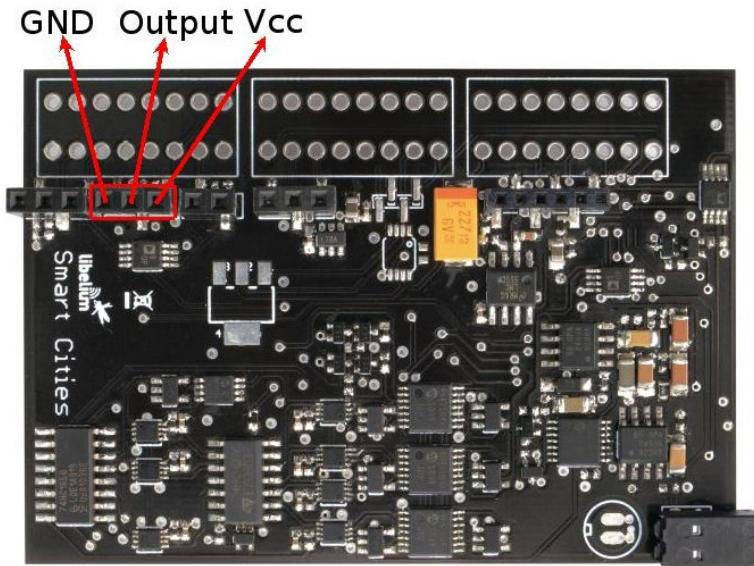


Figure 47: Image of the socket for connecting the MaxSonar® Sensors

## 4.7. Humidity Sensor (808H5V5)

### 4.7.1. Specifications

**Measurement range:** 0 ~ 100%RH  
**Output signal:** 0,8 ~ 3.9V (25°C)  
**Accuracy:** <±4%RH (at 25°C, range 30 ~ 80%), <±6%RH (range 0 ~ 100)  
**Typical consumption:** 0.38mA  
**Maximum consumption:** 0.5mA  
**Power supply:** 5VDC ±5%  
**Operation temperature:** -40 ~ +85°C  
**Storage temperature:** -55 ~ +125°C  
**Response time:** <15 seconds

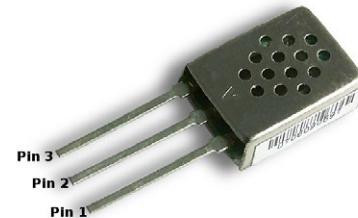


Figure 48: Image of the 808H5V5 sensor

### 4.7.2. Measurement Process

This is an analog sensor which provides a voltage output proportional to the relative humidity in the atmosphere. As the sensor's signal is outside of that permitted at the input of the analog-to-digital converter of the WaspMote's processor, its output voltage has been adapted to a range of values between 0,48 and 2,34V. In figure 49 we can see a graph of the output voltage vs the relative humidity prior to this conversion. The supply voltage of the sensor is controlled through a solid state switch, shared with the temperature, luminosity and dust sensors and activated with signal DIGITAL2.

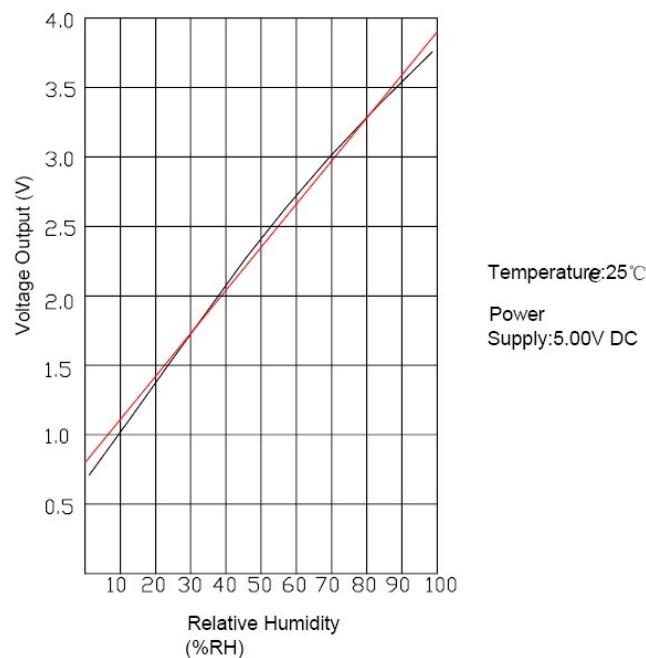


Figure 49: 808H5V5 humidity sensor output taken from the Sencera Co. Ltd sensor data sheet

Below a sample code for reading the output of the sensor and converting the voltage measured into relative humidity using the libraries of the board is shown:

```
{
  SensorCities.setBoardMode(SENS_ON);
  SensorCities.setSensorMode(SENS_CITIES_HUMIDITY, SENS_ON);
  delay(15000);
  float humidity_value;
  humidity_value = SensorCities.readValue(SENS_CITIES_HUMIDITY);
}
```

You can find a complete example code for reading the humidity in the following link:

<http://www.libelium.com/development/wasp mote/examples/sc-2-humidity-sensor-reading>

### 4.7.3. Socket

The socket of this sensor (2 ways, 2.54mm pitch) is shown in figure 50, in which the pin correspondence between them has been highlighted (pin 1 corresponds to Vcc, pin 2 corresponds to the output and pin 3 corresponds to GND). In section "Sockets for casing" more information about the corresponding pinout at the sockets for casing applications can be found.

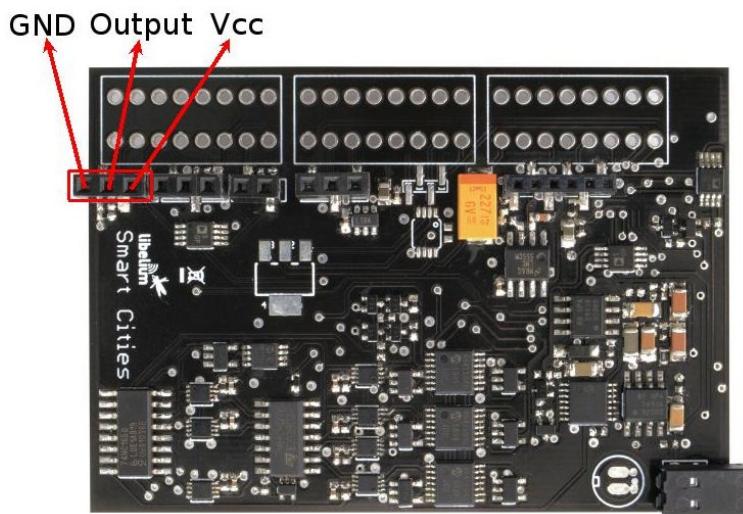


Figure 50: Image of the socket for the 808H5V5 sensor.

## 4.8. Temperature Sensor (MCP9700A)

### 4.8.1. Specifications

**Measurement range:** -40°C ~ +125°C

**Output voltage (0°C):** 500mV

**Sensitivity:** 10mV/°C

**Accuracy:** ±2°C (range 0°C ~ +70°C), ±4°C (range -40 ~ +125°C)

**Typical consumption:** 6µA

**Maximum consumption:** 12µA

**Power supply:** 2.3 ~ 5.5V

**Operation temperature:** -40 ~ +125°C

**Storage temperature:** -65 ~ 150°C

**Response time:** 1.65 seconds (63% of the response for a range from +30 to +125°C)



Figure 51: Image of the MCP9700A temperature sensor

### 4.8.2. Measurement Process

The MCP9700A is an analog sensor which converts a temperature value into a proportional analog voltage. The range of output voltages is between 100mV (-40°C) and 1.75V (125°C), resulting in a variation of 10mV/C°, with 500mV of output for 0°C. The output voltage may be directly captured by the analog-to-digital converter of the mote's processor in the input analog pin ANALOG4. The supply voltage of this sensor is controlled through the same switch that the sensors of humidity, luminosity and dust, activated by the digital pin DIGITAL2.

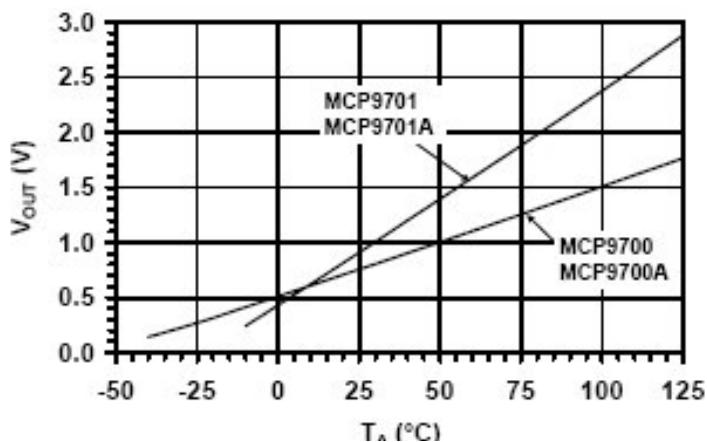


Figure 52: Graph of the MCP9700A sensor output voltage with respect to temperature, taken from the Microchip sensor's data sheet

A sample code for reading the sensor is provided below:

```
{
SensorCities.setBoardMode(SENS_ON);
SensorCities.setSensorMode(SENS_CITIES_TEMPERATURE, SENS_ON);
delay(100);
float temperature_value;
temperature_value = SensorCities.readValue(SENS_CITIES_TEMPERATURE);
}
```

You can find a complete example code for reading the temperature in the following link:

<http://www.libelium.com/development/wasp mote/examples/sc-1-temperature-sensor-reading>

### 4.8.3. Socket

The socket for the MCP9700A is shown in image 53, with the pin correspondence highlighted (Pin 1 corresponds to Vcc, pin 2 corresponds to the output and pin 3 corresponds to GND). More information about the socket for casing applications is shown in section "Sockets for casing".

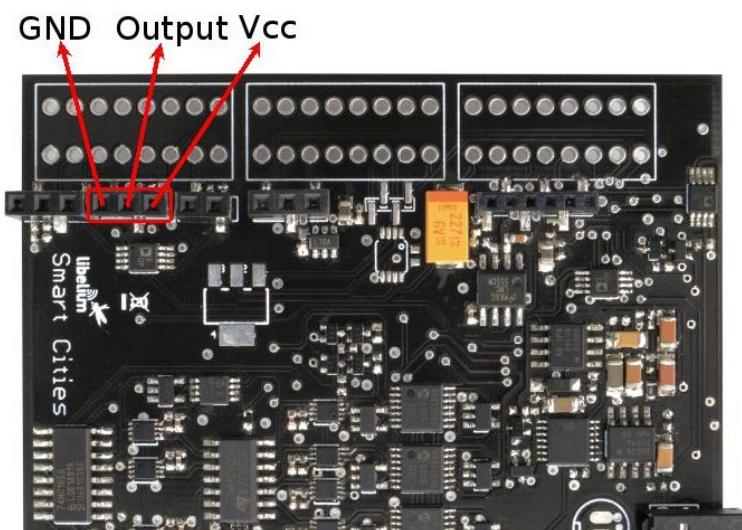


Figure 53: Image of the socket for the MCP9700A sensor

## 4.9. Luminosity Sensor (LDR)

### 4.9.1. Specifications

**Resistance in darkness:** 20MΩ

**Resistance in light (10lux):** 5 ~ 20kΩ

**Spectral range:** 400 ~ 700nm

**Operating Temperature:** -30°C ~ +75°C

**Minimum consumption:** 0uA approximately



Figure 54: Image of the LDR luminosity sensor

### 4.9.2. Measurement Process

This is a resistive sensor whose conductivity varies depending on the intensity of light received on its photosensitive part. The measurement of the sensor is carried out through the analog-to-digital converter of the mote's microcontroller, reading the resulting voltage out of a voltage divider formed by the sensor itself and the load resistor of the socket upon which it has been connected. This sensor shares the power supply with the dust, humidity and temperature sensors, which can be controlled through the output digital pin DIGITAL2, used to handle a solid state switch.

The measurable spectral range (400nm – 700nm) coincides with the human visible spectrum so it can be used to detect light/darkness in the same way that a human eye would detect it.

Below, a small sample of code for reading the output value of the sensor is shown:

```
{
  SensorCities.setBoardMode(SENS_ON);
  SensorCities.setSensorMode(SENS_CITIES_LDR, SENS_ON);
  delay(100);
  float ldr_value;
  ldr_value = SensorCities.readValue(SENS_CITIES_LDR);
}
```

You can find a complete example code for reading the LDR in the following link:

<http://www.libelium.com/development/wasp mote/examples/sc-3-ldr-sensor-reading>

### 4.9.3. Socket

In figure 55 we can see highlighted the socket upon which the LDR sensor must be placed. Since this sensor behaves as a simple resistor, polarity should not be taken into account when connecting it. More information about the sockets for casing applications can be found in section "Sockets for casing".

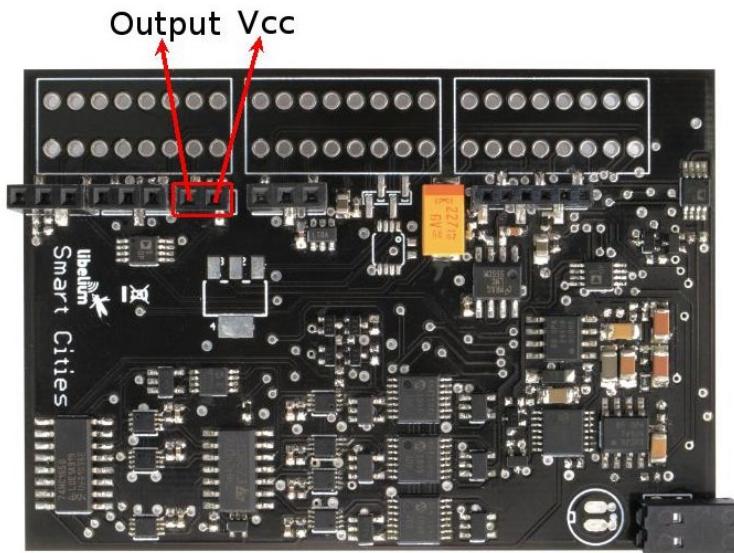


Figure 55: Image of the socket for the LDR sensor

## 4.10. Sensor interruptions

The Smart Cities board includes all the electronics necessary to generate interruptions from the output signals of all the sensors in it, so they can be used to manage associated processes in applications where a continuous monitoring of some of the parameters is required. It must be taken into account, regarding the calculation of the life of the battery, that not all the sensors in this board are low consumption devices, so it is possible that this monitoring will not be feasible unless a power source to recharge the battery continuously is available.

The interruptions from the sensors are generated in the board when one of those surpasses a threshold, defined with a digital potentiometer, setting the output of a voltage comparator high. Since there is only one interruption pin accessible from the board, the outputs of the comparators merge into a logic OR gate that generates a single output to trigger the interruption in the processor. A shift register captures the state of the comparators at the moment of the interruption, so it is not necessary to read all the sensors immediately afterwards to know which of them has triggered it.

In section "API" all the information necessary to manage the interruptions with the functions of the board's library is provided.

## 4.11. Sockets for casing

In case the Smart Cities board is going to be used in an application that requires the use of a casing, such as an outdoors application, a series of sockets to facilitate the connection of the sensors through a probe has been disposed.

These sockets (PTSM from Phoenix Contact) allow to assemble the wires of the probe simply by pressing them into it. To remove the wire press the above slot input pin and pull off the wire softly.

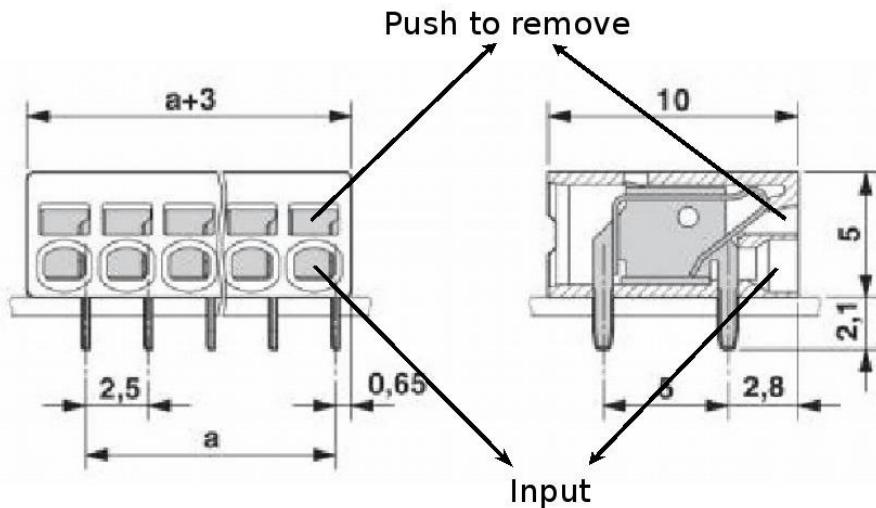


Figure 56: Diagram of a socket extracted from the Phoenix Contact data sheet

In the figure below an image of the board with the sockets in it and the correspondence between its inputs and the sensor's pins is shown.

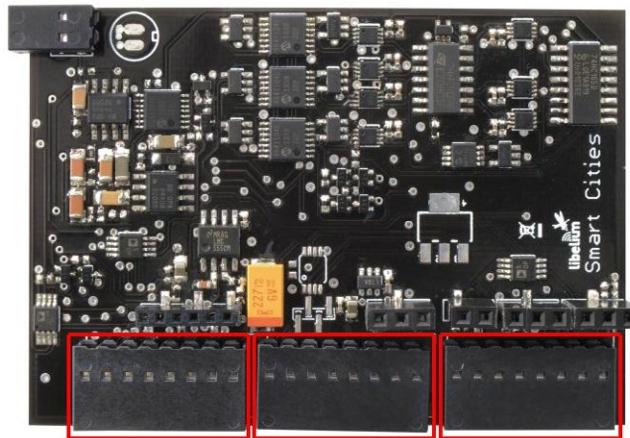


Figure 57: Image of the sockets for casing applications

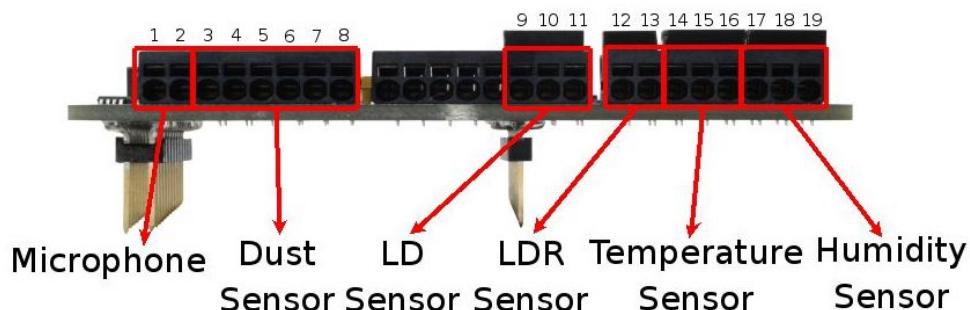


Figure 58: Image of the pin correspondence between the sockets and the sensors

Sensor	Pin	Function
<b>Microphone</b>	1	Output
	2	GND
	3	Vcc
	4	Vo
	5	S-GND
	6	LED
	7	LED-GND
	8	V-LED
<b>Particles sensor (PM-10) - Dust sensor</b>	9	Vcc
	10	Output
	11	GND
<b>LDR Sensor / Crack detection and propagation sensors</b>	12	Vcc
	13	Output
<b>Temperature sensor / Ultrasonic sensors</b>	14	Vcc
	15	Output
	16	GND
<b>Humidity sensor</b>	17	Vcc
	18	Output
	19	GND

## 5. Board configuration and programming

### 5.1. Hardware configuration

The Smart Cities board does not require of any handling of the hardware by the user except for placing the sensors in their corresponding position. In the section dedicated to each connector we can see an image of the connections between the socket and its corresponding sensor.

### 5.2. API

The Smart Cities Board library compiles a series of functions that allow the user to handle in an easy way all the resources of the board, the power supply, the start-up and reading of the sensors and the interruptions.

When using this board remember it is mandatory to include the SensorCities library by introducing the next line at the beginning of the code:

```
#include <WaspSensorCities.h>
```

The functions to handle all the features of the board, included in the WaspSensorCities API library, are detailed below:

#### **SensorCities.ON()**

Turns on the sensor board by activating the 3.3V and 5V supply lines.

#### **SensorCities.OFF()**

Turns off the sensor board by cutting the 3.3V and 5V supply lines.

#### **SensorCities.setBoardMode(MODE)**

This function is used to manage the power supply applied to the Smart Cities board. Assigning the value SENS\_ON to the variable MODE activates the WaspMote's switches which allow the passage of the 3.3V and 5V voltages, while assigning SENS\_OFF disconnects both switches cutting the power.

#### **SensorCities.setSensorMode(MODE, SENSOR)**

This function, similar to setBoardMode, allows to activate or deactivate the power of each sensor independently.

The state on which the sensor should be set is defined through the variable MODE, which can take the values SENS\_ON, to connect the power of the sensor, or SENS\_OFF, to disconnect it.

The sensor, circuit or group of sensors that we are going to manage is stored in the variable SENSOR, that can take the following values:

- SENS\_CITIES\_DUST, to activate the dust sensor.
- SENS\_CITIES\_CD, to activate the crack detection sensor.
- SENS\_CITIES\_CP, to activate the crack propagation sensor.
- SENS\_CITIES\_LD, to activate the linear displacement sensor.
- SENS\_CITIES\_AUDIO, to activate the microphone and its electronics.
- SENS\_CITIES\_ULTRASOUND\_3V3, to activate the ultrasonic sensor on the temperature socket.
- SENS\_CITIES\_AUDIO, to activate the microphone and its electronics.
- SENS\_CITIES\_ULTRASOUND\_5V, to activate the ultrasonic sensor on the humidity socket.
- SENS\_CITIES\_HUMIDITY, to activate the humidity sensor.
- SENS\_CITIES\_TEMPERATURE, to activate the temperature sensor.
- SENS\_CITIES\_LDR, to activate the LDR sensor.

As said in their respective sections, the DUST, LDR, TEMPERATURE and HUMIDITY sensors are controlled through the same switch, so when turning on or off one of them you will be acting on the whole group.

#### **SensorCities.readValue(SENSOR, TYPE)**

The function `readValue` may be used to execute the configuration, reading and conversion process of any of the sensors on the board through the analog-to-digital converter of the mote's processor. In the variable `SENSOR` the sensor to be read is introduced, and the output value is given in floating point format (type float). The values that can be assigned to that variable are:

- `SENS_CITIES_DUST`, to read the output of the dust sensor.
- `SENS_CITIES_CD`, to read the value of the crack detection sensor.
- `SENS_CITIES_CP`, to read the value of the crack propagation sensor.
- `SENS_CITIES_LD`, to read the value of the linear displacement sensor.
- `SENS_CITIES_AUDIO`, to read the output of the microphone.
- `SENS_CITIES_ULTRASOUND_3V3`, to read the output of the ultrasonic sensor on the temperature socket.
- `SENS_CITIES_HUMIDITY`, to read the output of the humidity sensor.
- `SENS_CITIES_ULTRASOUND_5V`, to read the output of the ultrasonic sensor on the humidity socket.
- `SENS_CITIES_TEMPERATURE`, to read the output of the humidity sensor.
- `SENS_CITIES_LDR`, to read the value of the LDR.

The parameter `TYPE` indicates the kind of ultrasound sensor when reading the any of the MaxBotix sensors. It is not necessary to introduce it when reading any other sensor of the board. The values that the `TYPE` parameter may take are:

- `SENS_US_WRA1`, to read the XL-MaxSonar®-WRA1 sensor.
- `SENS_US_EZ0`, to read the LV-MaxSonar®-EZ0 sensor.

#### **SensorCities.setThreshold(SENSOR, THRESHOLD)**

This function is used to configure the comparison threshold that regulates the interruption trigger from the Smart Cities Board. In the variable `SENSOR` the sensor whose comparison threshold is to be changed is introduced, the identifiers of the sensors mentioned before may be assigned to it. In the `THRESHOLD` variable the value to be given to this threshold is introduced in floating point format (float), which must be within a range between 0 and 3.3V.

#### **SensorCities.attachInt()**

The `attachInt` function, implemented as is in the code, including no parameters, enables interruptions generated by the board's sensors, allowing the microprocessor to recognize and process them as such.

#### **SensorCities.detachInt()**

Complementing the previous function, the aim of `detachInt` is to deactivate the interruptions if the microprocessor is not required to react in the event of a change in one of the sensors. After its execution the mote will ignore any interruption which arrives from the sensors until the `attachInt` instruction is activated again.

#### **SensorCities.loadInt()**

The `loadInt` instruction is used to read the content of the shift register and store its output in an integer variable called `SensorCities.intFlag`, in which the sensor that has caused the interruption and other sensors activated at that moment appear. Once all the registers have been read, they restart from zero, not loading again until a new interruption is triggered. To recognize if a sensor has produced an interruption, it is sufficient to carry out a logic comparison between the identifier of the sensor and the `intFlag` variable.

A basic program to detect events from the board will present a similar structure to the following, subject to changes in dependence of the application:

1. The board is switched on using the function `SensorCities.ON`.
2. Initialization of the RTC using `RTC.ON` to avoid conflicts in the I2C bus.
3. Configuration of the thresholds of those sensors which may generate an interruption with function `SensorCities.setThreshold`.

4. Activation of the sensors to generate given interruptions using function **SensorCities.setSensorMode**.
5. Enable interruptions from the board using the function **SensorCities.attachInt**.
6. Put the mote to sleep with the functions **PWR.sleep** or **PWR.deepSleep**.
7. When the mote wakes up, disable interruptions from the board using function **SensorCities.detachInt**.
8. Load the value stored in the shif register with function **SensorCities.loadInt**.
9. Process the interruption:
  - Turn on those inactive sensors to be read using function **SensorCities.setSensorMode**.
  - Take the measurements needed using function **SensorCities.readValue**.
  - Turn off the sensors that shall not generate an interrupt with function **SensorCities.setSensorMode**.
  - Store or send via a radio module the gathered information.
10. Return to step 5 to enable interruptions and put the mote to sleep.

Below is shown a sample code where the temperature, dust and linear displacement sensors are read every five minutes and their values sent through XBee 802.15.4. If an excessive audio value is detected, an alarm message is sent including the measurement from the microphone. In any case, if a crack has been detected, a message indicating it is included in every frame.

```
/* -----Smart Cities board example-----
www.Libelium.com
*/
// Inclusion of the Smart Cities Sensor Board library
#include <WaspSensorCities.h>

// Inclusion of the Frame library
#include <WaspFrame.h>

// Inclusion of the XBee 802.15.4 library
#include <WaspXBee802.h>

// Set interruption threshold
#define THRESHOLD 1.5

float audio_value = 0;
float crack_width_value = 0;
float temperature_value = 0;
float dust_value = 0;

// Pointer to an XBee packet structure
packetXBee* packet;

void setup()
{
    //Switch on the board
    SensorCities.ON();
    delay(100);

    // Init RTC
    RTC.ON();
    delay(100);

    // Configure not used interruption thresholds
    SensorCities.setThreshold(SENS_CITIES_LDR, 3.3);
    SensorCities.setThreshold(SENS_CITIES_DUST, 3.3);
    SensorCities.setThreshold(SENS_CITIES_LD, 3.3);
    SensorCities.setThreshold(SENS_CITIES_HUMIDITY, 3.3);
    SensorCities.setThreshold(SENS_CITIES_TEMPERATURE, 3.3);

    //Configure the threshold for the crack detection sensor
    SensorCities.setThreshold(SENS_CITIES_AUDIO, THRESHOLD);
```

```

        //Turn on the LDR
        SensorCities.setSensorMode(SENS_ON, SENS_CITIES_AUDIO);
        delay(3000);
    }
    void loop()
    {
        //Enable interruptions from the Cities Metering Board
        SensorCities.attachInt();

        //Put the mote to sleep
        PWR.deepSleep("00:00:00:10", RTC_OFFSET, RTC_ALM1_MODE1, UART0_OFF | UART1_OFF | BAT_OFF);

        //Disable interruptions from the sensor board
        SensorCities.detachInt();
        //Load the interruption register
        SensorCities.loadInt();

        // Create new frame (ASCII)
        frame.createFrame(ASCII,"WaspMote_Pro");

        SensorCities.setSensorMode(SENS_ON, SENS_CITIES_CD);
        delay(10);
        if( SensorCities.readValue( SENS_CITIES_CD ) == 0 )
        {
            // Add the alarm message to the composition
            frame.addSensor(SENSOR_STR, "New crack appeared!");
        }

        if ( SensorCities.intFlag & SENS_CITIES_AUDIO)
        {
            //Reading the temperature sensor
            audio_value = SensorCities.readValue(SENS_CITIES_AUDIO);
            frame.addSensor(SENSOR_MCP, audio_value);
        }
        else if ( intFlag & RTC_INT )
        {
            //Turn on the temperature sensor
            SensorCities.setSensorMode(SENS_ON, SENS_CITIES_TEMPERATURE);
            //Reading the temperature sensor
            temperature_value = SensorCities.readValue(SENS_CITIES_TEMPERATURE);
            // Turn off the temperature sensor
            SensorCities.setSensorMode(SENS_OFF, SENS_CITIES_TEMPERATURE);

            //Turn on the crack width sensor
            SensorCities.setSensorMode(SENS_ON, SENS_CITIES_LD);
            delay(100);
            //Reading the crack width sensor
            crack_width_value = SensorCities.readValue(SENS_CITIES_LD);
            //Turn off the crack width sensor
            SensorCities.setSensorMode(SENS_OFF, SENS_CITIES_LD);

            //Turn on the dust sensor
            SensorCities.setSensorMode(SENS_ON, SENS_CITIES_DUST);
            delay(2000);
            //Reading the crack width sensor
            dust_value = SensorCities.readValue(SENS_CITIES_DUST);
            //Turn off the crack width sensor
            SensorCities.setSensorMode(SENS_OFF, SENS_CITIES_DUST);

            // Add the values read to the frame composition
            frame.addSensor(SENSOR_TCA, temperature_value);
            frame.addSensor(SENSOR_LD, crack_width_value);
            frame.addSensor(SENSOR_DUST, dust_value);
        }
    }

```

```
// Init XBee
xbee802.ON();
// Set parameters to packet:
packet=(packetXBee*) calloc(1,sizeof(packetXBee));
packet->mode=BR0ADCAST;

// Set destination XBee parameters to packet
xbee802.setDestinationParams( packet, "000000000000FFFF", frame.buffer, frame.length);

// Send XBee packet
xbee802.sendXBee(packet);

// Turn off the XBee Module
xbee802.OFF();
delay(100);

// Clear the interruption flag
clearIntFlag();

}
```

The files related to this sensor board are: `WaspSensorCities.cpp`, `WaspSensorCities.h`

They can be downloaded from:

[http://www.libelium.com/development/wasp mote/sdk\\_and\\_applications](http://www.libelium.com/development/wasp mote/sdk_and_applications)

## 6. Consumption

### 6.1. Power control

The Smart Cities Board for WaspMote requires both the 3.3V and 5V power supplies output from the mote.

The sensors are powered through three solid state switches that allow to cut or activate separately the supply voltages for each sensor or group of sensors.

The microphone and the linear displacement sensor can be turned on separately, while the power supply of the humidity, luminosity, temperature and dust sensors is controlled through a switch shared by all of them.

Thus, DIGITAL6 pin controls the switch that activates the 3.3V supply for the microphone, the DIGITAL7 pin controls the linear displacement sensor's switch (also powered at 5V) and the DIGITAL2 pin controls the supply voltages of the remaining group of sensors (dust, humidity, luminosity and temperature), which needs both 3.3V and 5V supplies.

All these switches may be controlled through the `SensorCities.setBoardMode` and `SensorCities.setSensorMode` functions implemented in the API. You can find more information about it in section "API".

### 6.2. Tables of consumption

In the following table the consumption of the board is shown, the constant minimum consumption (fixed by the permanently active components) and the consumption of each of the independent blocks that may be powered independently. The board's power can be completely disconnected, reducing the consumption to zero, using the 3.3V and the 5V main switches disconnection `SensorCities.setBoardMode` command included in the library.

Consumption	
<b>Minimum (Constant)</b>	400µA
<b>Various Sensors Group (no sensors connected)</b>	12.5mA
<b>Various Sensors Group (only temperature sensor connected)</b>	12.5mA
<b>Various Sensors Group (only humidity sensor connected)</b>	13.2mA
<b>Various Sensors Group (only luminosity sensor connected)</b>	12.8mA
<b>Various Sensors Group (only ultrasonic sensor connected)</b>	15mA
<b>Crack detection sensor</b>	500µA
<b>Crack propagation sensor</b>	35mA
<b>Various Sensors Group (only dust sensor connected)</b>	37.5mA
<b>Linear displacement sensor</b>	8.3mA
<b>Microphone</b>	1.1mA

## 6.3. Low consumption mode

The Smart Cities Board has been designed to minimize the consumption of the mote in operation conditions as long as in low consumption modes.

- **Avoid activating all the sensors at the same time**

Although there are no high consumption sensors in the Smart Cities board, increasing the load in the 5V line implies an increase in the DC-DC voltage converter in the mote, so it is recommended, if possible, to avoid activating the sensors powered at 5V at the same time.

- **Use the WaspMote low consumption mode**

As the other sensor boards for WaspMote, the library of the Smart Cities Board includes all the functions needed to deactivate the sensors which are not being used and put the mote in low consumption mode.

- **Do not connect senors that are not going to be used**

Since several sensors share the same power line, a sensor that is not going to be used connected to the board will entail an additional consumption, and so a shorter life of the battery.

## 7. API Changelog

Function / File	Changelog	Version
#include	Remember to include the WaspSensorParking library in the top of your pde	V0.31 → v1.0
SensorCities.ON()	New function to turn on the board	V0.31 → v1.0
SensorCities.OFF()	New function to turn off the board	V0.31 → v1.0
SensorCities.readValue	Added the possibility of including the sensor type when reading the ultrasound sensor	V0.31 → v1.0
SensorCities.setAudioGain	This function is no longer available, since it is not necessary anymore	V0.31 → v1.0
SENS_NOISE_ADDRESS	Though it does not affect regular programming, it must be taken into account that the EEPROM addresses of the calibration coefficients have changed	V0.31 → v1.0

## 8. Documentation changelog

- Added references to 3G/GPRS Board in section: Radio Interfaces.

## 9. Maintenance

- In this section, the term "Waspmote" encompasses both the Waspmote device itself as well as its modules and sensor boards.
- Take care with the handling of Waspmote, do not drop it, bang it or move it sharply.
- Avoid putting the devices in areas of high temperatures since the electronic components may be damaged.
- The antennas are lightly threaded to the connector; do not force them as this could damage the connectors.
- Do not use any type of paint for the device, which may damage the functioning of the connections and closure mechanisms.

## 10. Disposal and recycling

- In this section, the term "Wasp mote" encompasses both the Wasp mote device itself as well as its modules and sensor boards.
- When Wasp mote reaches the end of its useful life, it must be taken to a recycling point for electronic equipment.
- The equipment has to be disposed on a selective waste collection system, different to that of urban solid waste. Please, dispose it properly.
- Your distributor will inform you about the most appropriate and environmentally friendly waste process for the used product and its packaging.

