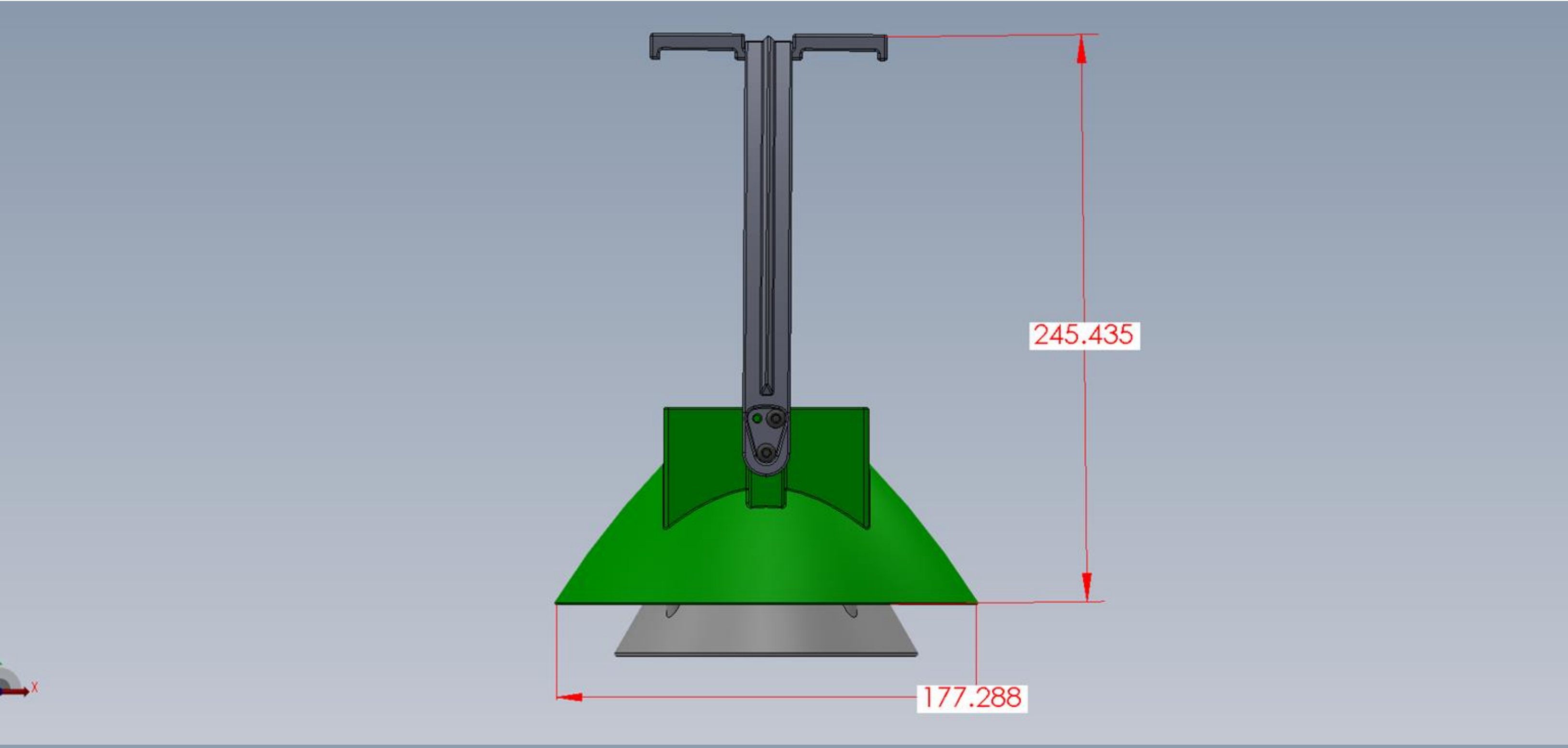
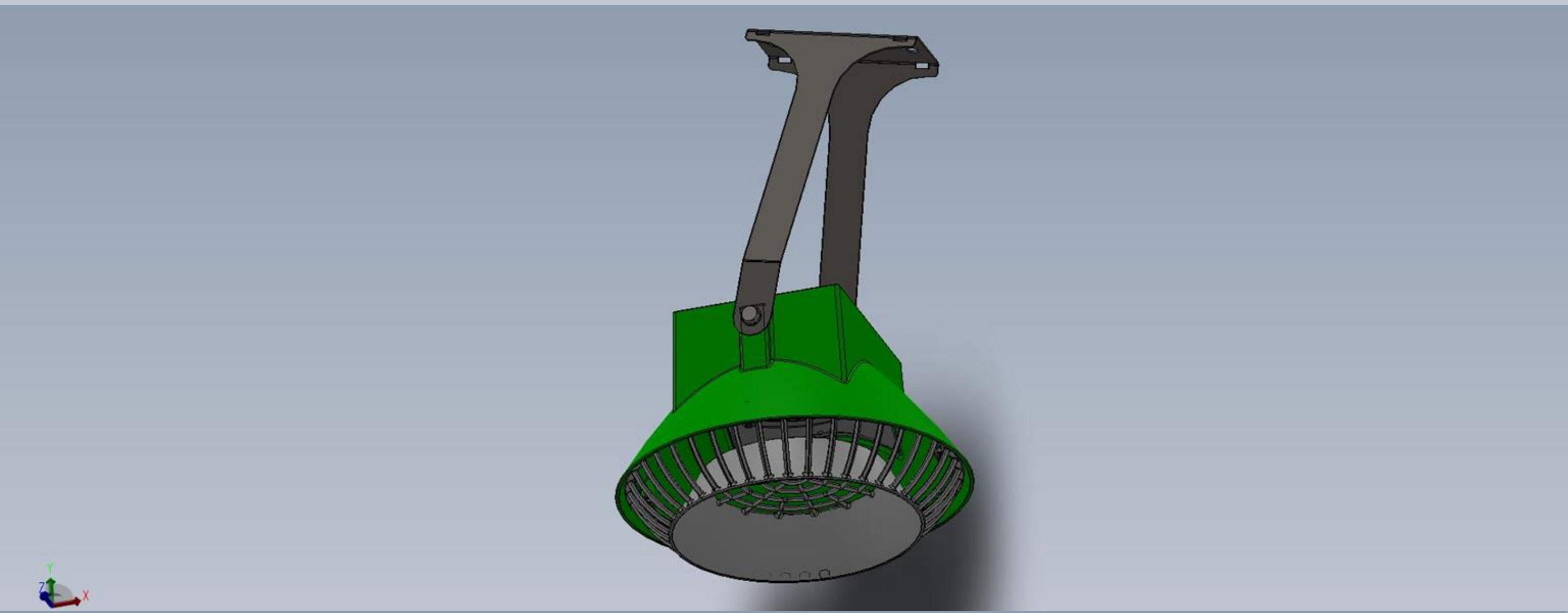
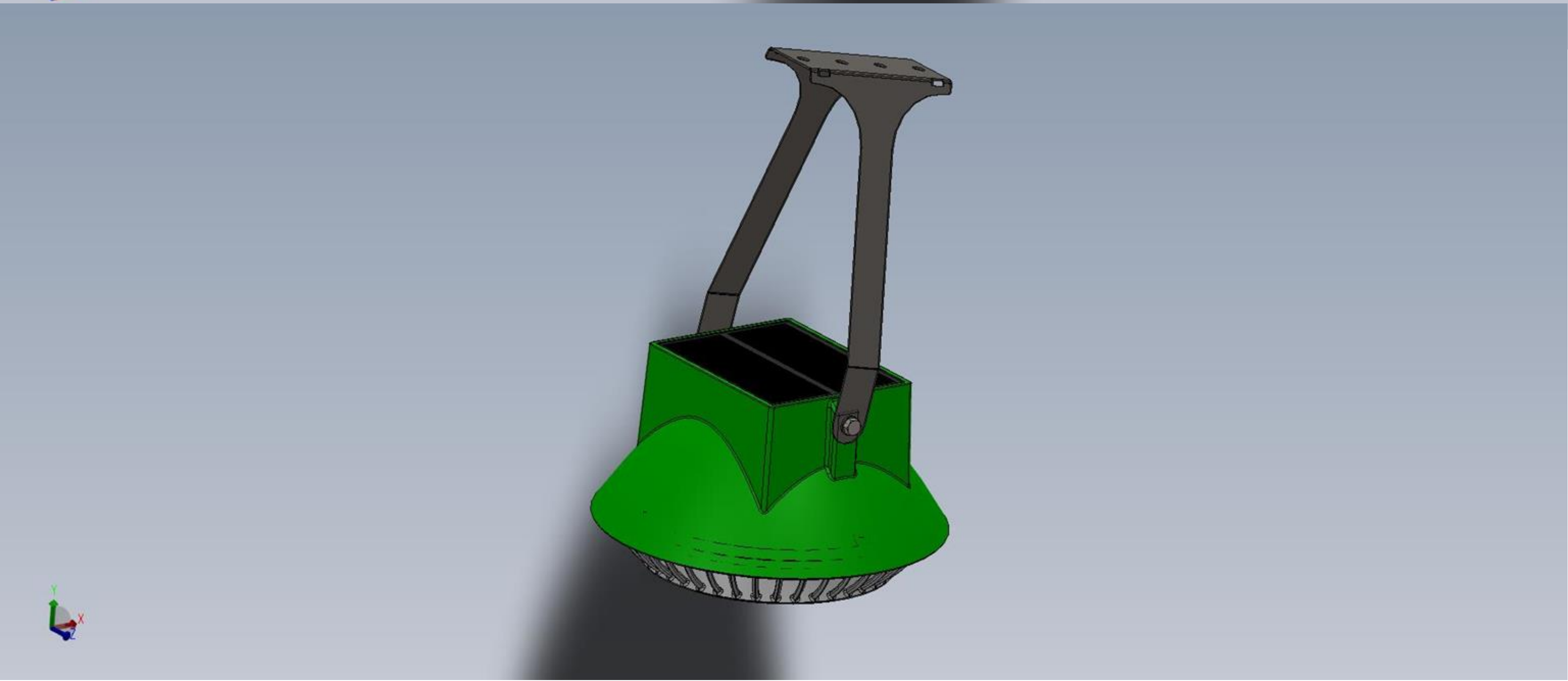
Early wildfire sensor specs model
WS02revC01

LAD Sensors

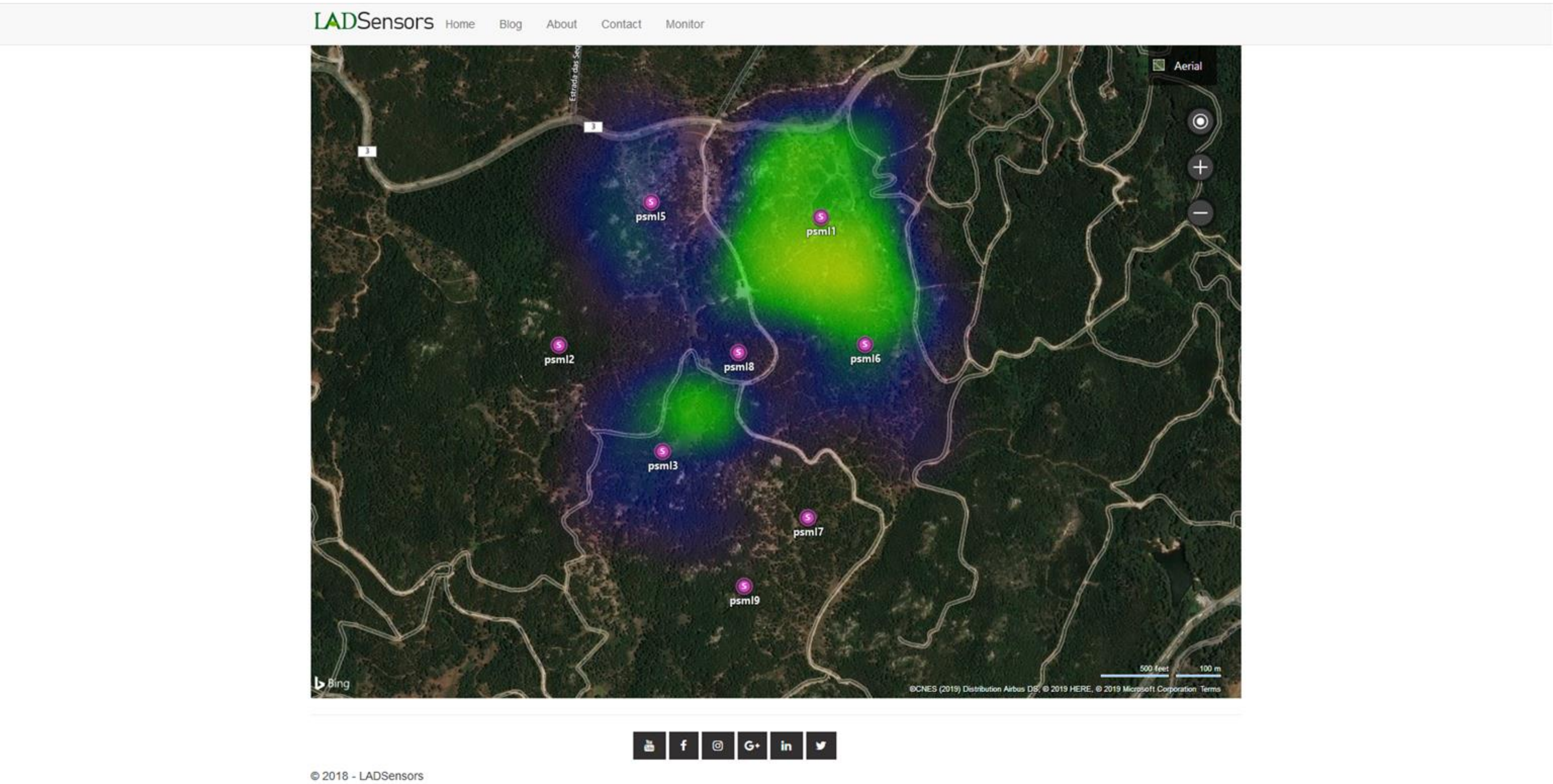


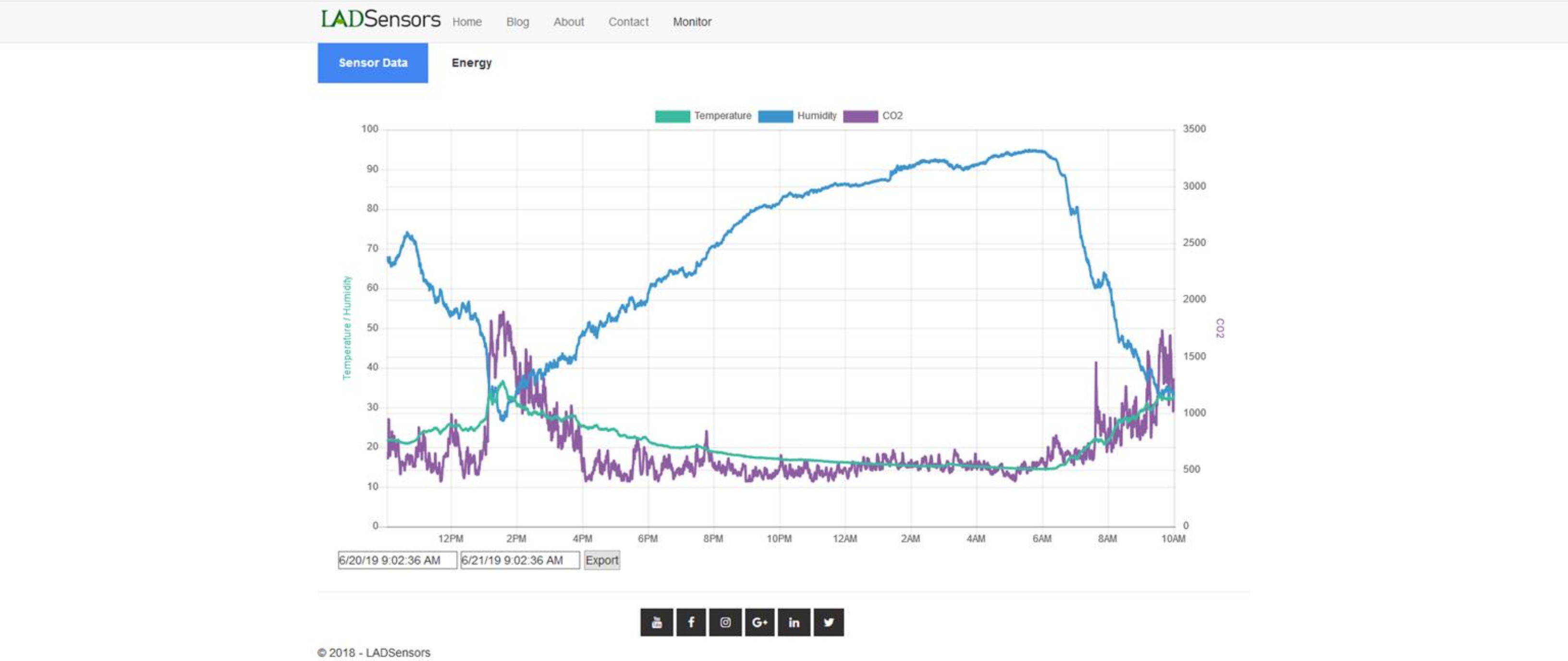


245.435
177.288





Real-time wildfire monitoring, detection and prediction
Cost effective and fast early wildfire detection in less than 4 min.





Environmental monitoring, enabling fauna and flora protection

Unlike conventional batteries that can leak chemicals to it's surroundings over time, our patented energy storage unit is safer withstanding higher temperature ranges as well environment friendly.

- Sensors are fully energy autonomous using embedded solar panels and no batteries.
- Sensors do not require assistance of any kind of batteries.
- Solar panel is able to charge its electrical circuitry even under heavy cloudy environment.
- Data communication is based on LoRa® technology according to EU regulations.
- IP65 certification (dust, rain and humidity resistible).
- The sensor is a unified single body containing the sensor, the respective electrical circuitry, the solar panel and the mounting strap mechanism.

Sensor Specification

1.1 General Electrical Specification

Table 1: Electrical parameter specification

Operating Conditions						
Parameter	**Symbol**	**Condition**	**Min**	**Typ**	**Max**	**Unit**
Supply Voltage Internal Domains	V_{DD}	ripple max. 50 mVpp	1.71	1.8	3.6	V
Supply Voltage I/O Domain	V_{DDIO}		1.2	1.6	3.6	V
Sleep current	I_{DDSL}			0.15	1	µA
Standby current (inactive period of normal mode)	I_{DDSB}			0.29	0.8	µA
Current during humidity measurement	I_{DDH}	Max value at 85 °C		340	450	µA
Current during pressure measurement	I_{DDP}	Max value at -40 °C		714	849	µA
Current during temperature measurement	I_{DDT}	Max value at 85 °C		350		µA
Start-up time	$t_{startup}$	Time to first communication after both V_{DD} > 1.58 V and V_{DDIO} > 0.65 V			2	ms
Power supply rejection ratio (DC)	PSRR	full V_{DD} range			±0.01 ±5	%r.H./V Pa/V
Standby time accuracy	$\Delta t_{standby}$			±5	±25	%

Table 2: Gas sensor parameter specification

Parameter	Symbol	Condition	Min	Typ	Max	Unit
Operational range			-40		85	°C
			0		100	% r.H.
Supply Current during heater operation	I_{DD}	Heater target temperature 320 °C, constant operation ($V_{DD} \leq 1.8$ V, 25°C)	9	12	13	mA
Peak Supply Current	I_{Peak}	Occurs within first ms of switching on the hotplate	15	17	18	mA
Average Supply Current ($V_{DD} \leq 1.8$ V, 25°C)	$I_{DD,IAQ}$	Ultra-low power mode		0.09		mA
		Low power mode		0.9		mA
		Continuous mode		12		mA
Response time (brand-new sensors)	$\tau_{33-63\%}$	Ultra-low power mode		92		s
	$\tau_{33-63\%}$	Low power mode		1.4		s
	$\tau_{33-63\%}$	Continuous mode		0.75		s
Resolution of gas sensor resistance measurement			0.05	0.08	0.11	%
Noise in gas sensor resistance (RMS)	N_R			1.5		%

Table 3: bVOC mixture with Nitrogen as carrier gas

Molar fraction	Compound	Production tolerance	Certified accuracy
5 ppm	Ethane	20 %	5 %
10 ppm	Isoprene /2-methyl-1,3 Butadiene	20 %	5 %
10 ppm	Ethanol	20 %	5 %
50 ppm	Acetone	20 %	5 %
15 ppm	Carbon Monoxide	10 %	2 %

1.2 Humidity sensor specification

Table 4: Humidity parameter specification

Parameter	Symbol	Condition	Min	Typ	Max	Unit
Operating Range			-40	25	85	°C
			0		100	% r.H.
Full accuracy range			0		65	°C
			10		90	% r.H.
Supply Current	$I_{DD,H}$	1 Hz forced mode, temperature and humidity measurement		2.1	2.8	µA
Absolute Accuracy	A_H	20–80 % r.H., 25 °C, including hysteresis		±3		% r.H.
Hysteresis	H_H	10→90→10 % r.H., 25°C		±1.5		% r.H.
Nonlinearity	NL_H	10→90 % r.H., 25°C		1.7		% r.H.
Response time to complete 63% of step	$\tau_{0-63\%}$	N_2 (dry) → 90 % r.H., 25°C		8		s
Resolution	R_H			0.008		% r.H.
Noise in humidity (RMS)	N_H	Highest oversampling		0.01		% r.H.
Long-term stability	ΔH_{stab}	10–90 % r.H., 25°C		0.5		% r.H./ year

1.3 Pressure sensor specification

Table 5: Pressure parameter specification

Parameter	Symbol	Condition	Min	Typ	Max	Unit
Operating temperature range	T_A	operational	-40	25	85	°C
		full accuracy	0		65	
Operating pressure range	P	full accuracy	300		1100	hPa
Supply current	$I_{DD,LP}$	1 Hz forced mode, pressure and temperature, lowest power		3.1	4.2	µA
Temperature coefficient of offset	TCO_P	25–40 °C, 900 hPa		±1.3		Pa/K
				±10.9		cm/K
Absolute accuracy pressure	$A_{p, full}$	300–1100 hPa 0–65°C		±0.6		hPa
Relative accuracy pressure	A_{rel}	700–900hPa, 25–40 °C, at constant humidity		±0.12		hPa
	A_{rel}	900–1100hPa 25–40 °C, at constant humidity		±0.12		hPa
Resolution of pressure output data	R_P	Highest oversampling		0.18		Pa
Noise in pressure	$N_{P,fullBW}$	Full bandwidth, highest oversampling		1.4		Pa
				11		cm
		Reduced bandwidth, highest oversampling		0.2		Pa
				1.7		cm
Solder drift		Minimum solder height 50µm	-0.5	1.2	+2.0	hPa
Long-term stability	P_{stab}	per year		±1.0		hPa
Possible sampling rate	f_{sample_P}	Lowest oversampling, see chapter 3.3.2	157	182		Hz

1.4 Temperature sensor specification

Table 6: Temperature parameter specification

Parameter	Symbol	Condition	Min	Typ	Max	Unit
Operating temperature range	T_A	operational	-40	25	85	°C
Supply current	$I_{DD,T}$	1 Hz forced mode, temperature measurement only		1.0		µA
Absolute accuracy temperature	$A_{T,25}$	25 °C		±0.5		°C
	$A_{T,full}$	0–65 °C		±1.0		°C
Output resolution	R_T	API output resolution		0.01		°C
RMS noise	N_T	Lowest oversampling		0.005		°C

Table 7: General sensor specifications

Parameter	Condition	Min	Max	Unit
Operating temperature range	operational	-40	85	°C
Operating humidity range	operational	0	100	%
Weight	Weight (including the sensor and the strap mechanism that helps mounting sensors on trees)		450	g
Autonomy	Operational autonomy based on interval of 90 seconds in absolute dark environment	36	64	H
Data acquisition	Interval of data acquisition, including temperature, humidity, atmospheric pressure and carbon dioxide (CO2)	3	3600	s
Data transmission	Interval of data transmission of acquired data including temperature, humidity, atmospheric pressure and carbon dioxide (CO2)	10	3600	s

Sensor calibration

There is no need for a calibration procedure. CO2 sensor is calibrated automatically and maintained automatically in software. In order for the sensor to calibrate correctly, it needs to be exposed to clean air for at least 24 hours. In those 24 hours CO2 readings are valid, but accuracy is not in it’s fullest.

Product Lifetime > 5 years

2. LoRaWAN Specifications

Table 8: Wireless parameters

Working frequency	863MHz ~ 870MHz 902MHz ~ 928MHz
Protocol	LoRaWAN
Maximum Tx power	19 dbm
Rx Sensitivity	-140 dbm
Communication distance	Up to 15 km (LOS)

Communication

WS02revA01 sensor uses a custom Bit Tag Value message blocks in order to send/receive data. Ambient values may not always be present in the message block, if they are not present user can assume that current value of this missing tag is the same from last message. This is done in order to save message length and shorten communication time.

Bit Tag name	Bit	Tag Value Type	Tag Value Size	Value Units
Battery Level	1	byte	1	Percentage
CO2	2	ushort	2	Parts per million
Temperature	3	float	4	Degrees Celsius
Humidity	4	float	4	Relative Percentage
Atmospheric Pressure	5	float	4	Hectopascal (hPa)
Raw Gas Resistance	6	uint	4	Ohm's
Timestamp	7	ushort	2	Milliseconds

Message format:

{Bit info Block} {Value} {Value} {Value}… {Bit info Block} {Value} {Value} {Value}

C# parsing code example:

```
[Flags]
public enum FSSensorPresentDeviceData
{
   None = 0,
   BatteryLevel = 1,
   Co2 = 2,
   Temperature = 4,
   Humidity = 8,
   AtmosphericPressure = 16,
   RawGasResistance = 32,
   Timestamp = 64
}

public static List<FSSensorData> Parse(byte[] data, DateTime payloadTime)
{
   List<FSSensorData> result = new List<FSSensorData>();

   using (MemoryStream deviceDataStream = new MemoryStream(data))
   using (BinaryReader binaryReader = new BinaryReader(deviceDataStream))
   {
      while (deviceDataStream.Position < deviceDataStream.Length)
      {
         FSSensorPresentDeviceData dataPresent = (FSSensorPresentDeviceData)binaryReader.ReadByte();

         if ((dataPresent & FSSensorPresentDeviceData.BatteryLevel) != 0)
         {
            var val = binaryReader.ReadByte();
            smd.BatteryLevel = (ushort)(val & 0x7F);
            if ((val & (1 << 7)) != 0)
            {
               val = binaryReader.ReadByte();
               smd.BatterySourceLevel = (ushort)(val & 0x7F);
            }
         }

         if ((dataPresent & FSSensorPresentDeviceData.Co2) != 0) smd.CO2 = binaryReader.ReadUInt16();
         if ((dataPresent & FSSensorPresentDeviceData.Temperature) != 0) smd.Temperature = binaryReader.ReadSingle();
         if ((dataPresent & FSSensorPresentDeviceData.Humidity) != 0) smd.Humidity = binaryReader.ReadSingle();
         if ((dataPresent & FSSensorPresentDeviceData.AtmosphericPressure) != 0) smd.AtmosphericPressure = binaryReader.ReadSingle();
         if ((dataPresent & FSSensorPresentDeviceData.RawGasResistance) != 0) smd.RawGasResistance = binaryReader.ReadUInt32();
         if ((dataPresent & FSSensorPresentDeviceData.Timestamp) != 0)
         {
            var ts = binaryReader.ReadUInt16();
            smd.Timestamp = payloadTime.Subtract(new TimeSpan(0, 0, ts));
         }

         result.Add(smd);
      }
   }

   var mdSize = result.Count;
   for (int i = 0; i < mdSize; i++)
   {
      if (result[i].Timestamp == DateTime.MinValue)
      {
         result[i].Timestamp = payloadTime.Subtract(new TimeSpan(0, 0, (mdSize - 1 - i) * 30));
      }
   }

   return result;
}
```

Javascript parsing code example:

```
const FSSensorPresentDeviceData =
{
	None : 0,
	BatteryLevel : 1,
	Co2 : 2,
	Temperature : 4,
	Humidity : 8,
	AtmosphericPressure : 16,
	RawGasResistance : 32,
	Timestamp : 64
}

//var data = "eV57lLxCj4JyRDg/DAAcADlc16O8QhR+ckSkVwwA";

let buff = Buffer.from(data, 'base64');
var buffLen = buff.length;

var position = 0;
var datas = [];
while(position < buffLen)
{
  var smd = {};

  var dataPresent = buff.readUInt8(position++);
  console.log(dataPresent);

  if ((dataPresent & FSSensorPresentDeviceData.BatteryLevel) !== 0)
  {
    var val = buff.readUInt8(position++);
    smd.BatteryLevel = (val & 0x7F);
    console.log("BatteryLevel = " + smd.BatteryLevel);
    if ((val & (1 << 7)) !== 0)
    {
      val = buff.readUInt8(position++);
      smd.BatterySourceLevel = (val & 0x7F);
      console.log("BatterySourceLevel = " + smd.BatterySourceLevel);
    }
  }

  if ((dataPresent & FSSensorPresentDeviceData.Co2) !== 0)
  {
    smd.CO2 = buff.readUInt16LE(position);
    position = position + 2;
    console.log("CO2 = " +smd.CO2);
  }
  if ((dataPresent & FSSensorPresentDeviceData.Temperature) !== 0)
  {
    smd.Temperature = buff.readFloatLE(position);
    position = position + 4;
    console.log("Temperature = " + smd.Temperature);
  }
  if ((dataPresent & FSSensorPresentDeviceData.Humidity) !== 0)
  {
    smd.Humidity = buff.readFloatLE(position);
    position = position + 4;
    console.log("Humidity = " + smd.Humidity);
  }
  if ((dataPresent & FSSensorPresentDeviceData.AtmosphericPressure) !== 0)
  {
    smd.AtmosphericPressure = buff.readFloatLE(position);
    position = position + 4;
    console.log("AtmosphericPressure = " + smd.AtmosphericPressure);
  }
  if ((dataPresent & FSSensorPresentDeviceData.RawGasResistance) !== 0)
  {
    smd.RawGasResistance = buff.readUInt32LE(position);
    position = position + 4;
    console.log("RawGasResistance = " + smd.RawGasResistance);
  }
  if ((dataPresent & FSSensorPresentDeviceData.Timestamp) !== 0)
  {
    smd.ts = buff.readUInt16BE();
    position = position + 2;
    console.log("ts = " + smd.ts);
  }
```

```
  datas.push(smd);
}
```

Note : In order to save message size, last block time can be assumed it is current time, and blocks before are the subtracted seconds from current time. Assuming message was received at 13:46:37, then first block time is 13:46:07 (13:46:37 - 30 seconds (in case timestamp is 30 seconds)).

Data examples (base64):

Example 1:

`aDGIi0KCmQMAHgA8mpmnQUx3i0I9in1EjJcDAA==`

```
{
   Humidity: 69.76599884033203,
   RawGasResistance: 235906,
   ts: 26673
}
,
{
  Temperature: 20.950000762939453,
   Humidity: 69.73300170898438,
   AtmosphericPressure: 1014.1599731445312,
   RawGasResistance: 235404
}
```


Example 2:

`feRYMzOtQVI4kkLhun1EzhADAB4AOeRXmtmRQlK4fUQkCwMA==`

```
{
  BatteryLevel: 100,
   BatterySourceLevel: 88,
   Temperature: 21.649999618530273,
   Humidity: 73.11000061035156,
   AtmosphericPressure: 1014.9199829101562,
   RawGasResistance: 200910,
   ts: 32228 }
,
{
   BatteryLevel: 100,
   BatterySourceLevel: 87,
   Humidity: 72.92500305175781,
   AtmosphericPressure: 1014.8800048828125,
   RawGasResistance: 199460
}
```

Example 3:

TeRXpHCtQXd+kkIeACnkWOzRkUKQBQMA

{
  BatteryLevel: 100,
  BatterySourceLevel: 87,
  Temperature: 21.68000030517578,
  Humidity: 73.24700164794922,
  ts: 19940
}
,
{
  BatteryLevel: 100,
  BatterySourceLevel: 88,
  Humidity: 72.91000366210938,
  RawGasResistance: 198032
}

Example 4:

XeRXH4WtQeE6kkJSuH1EHgAp5FjPt5FCBgYDAA==

{
  BatteryLevel: 100,
  BatterySourceLevel: 87,
  Temperature: 21.690000534057617,
  Humidity: 73.11499786376953,
  AtmosphericPressure: 1014.8800048828125,
  ts: 24036
}
,
{
  BatteryLevel: 100,   BatterySourceLevel: 88,
  Humidity: 72.85900115966797,
  RawGasResistance: 198150
}