Remote

Solar Powered

(with battery backup)

Soil Moisture Sensor

Notes

# Purpose

This device will periodically send a value representing the moisture in soil in the range of completely dry to completely wet.

Barring weathering and corrosion, it is expected that it will require no maintenance for many years. (eg. no battery replacement)

# How it works

The [RFNano](https://www.aliexpress.us/item/3256807986944972.html)[[1]](#footnote-0) wakes up from sleep every hour, measures the soil moisture, and transmits the value to a [MySensors Gateway](https://www.mysensors.org/build/select_gateway).

How can this be difficult? The [soil moisture sensor](https://www.amazon.com/Capacitive-Soil-Moisture-Sensor-Electronic/dp/B0B8N69LMX/ref=sr_1_1) is connected to power, ground and an analog input of the Nano. The value is read with a single command, analogRead(An); which returns an integer value. The map(); function makes a number between 0 and 100. The MySensors send(); command sends it to the gateway. And finally the MySensors sleep(); function puts the unit into a low(ish) power state for the specified time.

# The first challenge

## The first challenge of the first challenge

The challenge starts with the [map();](https://docs.arduino.cc/language-reference/en/functions/math/map/) function. (Please see reference.) What is the value read when the sensor is absolutely dry? And the value when immersed in water (up to the line on the sensor)? It turns out this value is different for every sensor and every Arduino. So each sensor/Arduino pair has to have these values be measured and stored some place.

But it’s not as simple as that, the wet and dry values change depending on the supply voltage. The supply voltage is determined by the state of charge of the battery and the characteristics of the battery charge controller. And the relationship between the values and supply voltage is non-linear at the low voltage end.

It was decided that non-linearity would be ignored by making the calibration measurements in the linear region with the rationale that when the soil gets that dry, watering needs to be done soon.

So four calibration measurements are required: dry-maxVolts (CxD), dry-minVolts (CnD), wet-maxVolts (CxW) and wet-minVolts (CnW).

This will require that the supply voltage Voltage be read at the same time the moisture sensor is read. Thus, a two level map function:

map(Count, map(Voltage, Vbat\_MAX,Vbat\_MIN, CxD,CnD), map(Voltage, Vbat\_MAX,Vbat\_MIN, CxW,CnW), 0, 100);

(line 379 of the code *MySensorsRemoteSolarMoistureSensor\_5\_02)*

## The second challenge of the first challenge, where to store the calibration data.

Three choices were considered.

* Hard code
* Externally
* EEPROM of the Arduino

The first choice was rejected because it would require code maintenance for each soil moisture sensor.

The second choice was rejected because it would require more code gymnastics than the third option.

The third option requires a second calibration program or additional code in the main program. The latter was chosen because only one Arduino program would have to be maintained, though this introduced two additional challenges:

* It would require coding in the host controller. (The host controller I chose was Home Assistant, which has a whole set of challenges in and of itself. These will be described later.)
* The EEPROM stores data in individual bytes and an integer is two bytes.

### The calibration procedure

1. The device is prepared for calibration and set to condition 1.
   * 1. The battery/solar panel are removed and replaced with variable power supply
     2. The supply is set to maximum voltage, 4.2V
     3. The sensor is wiped dry
   1. The Controller (eg. Home Assistant) is instructed to send the device a value of 1 for the variable **calnnn**.
   2. The device will measure the current supply voltage and store it EEPROM as Vbat\_MAX
   3. The device will store the current analog value in EEPROM as “dry-maxVolts (CxD)”
2. The device is prepared for calibration and set to condition 2.
   * 1. The voltage is set to minimum voltage, 3.3V
   1. The Controller is instructed to send the device a value of 2 for the variable **calnnn**.
   2. The device will measure the current supply voltage and store it EEPROM as Vbat\_MIN
   3. The device will store the current analog value in EEPROM as “dry-minVolts (CnD)”
3. The device is prepared for calibration and set to condition 3.
   * 1. The sensor of the device is immersed in water up to the line.
     2. The device will sample the sensor every 2 seconds and send the value to the controller
     3. When it is observed that the value has stabilized (about 2 minutes), the Controller is instructed to send the device a value of 2 for the variable **calnnn**.
   1. The Controller is instructed to send the device a value of 3 for the variable **calnnn**.
   2. The device will store the current analog value in EEPROM as “wet-minVolts (CnW)”
4. The device is prepared for calibration and set to condition 4.
   * 1. The voltage is set to minimum voltage, 4.2V
   1. The Controller is instructed to send the device a value of 4 for the variable **calnnn**.
   2. The device will store the current analog value in EEPROM as “wet-maxVolts (CxW)”

# Second Challenge: Communication between the Arduino and Home Assistant

## Sending data to Home Assistant

There are three functions that send data to the host, sendSketchInfo(), present(), and send() . It is possible to send data too fast to Home Assistant. The wait()[[2]](#footnote-1) function needs to be employed after present(), and send(). The time to wait is dependent on the type of gateway. In the case of an MQTT gateway 4000 and 500 milliseconds respectively. With a serial gateway, 500 and 200.

In order for Home Assistant to recognize that a variable functionally exists, a value has to be sent, too. Just presenting is insufficient.

## Receiving data from Home Assistant

### Asynchronous receiving

Data is received by a MySensors background process and only when the device is not “sleeping” The MySensors’ receive() routine, as with any interrupt service routine (ISR), requires the time spent be minimized to the extreme. Basically, capture the data and set a flag that indicates that there is data available.

Because the data can come in at any time the variables used must be designated as volatile and be global variables It’s hard to explain … each routine sets up its own environment and the variables won’t change values unless expressly stated to do so. In some cases, like in loop() the ISR needs to change the value. Thus volatile.

Therefore, loop() and other routines need to check flags set in receive() and if the flag set, operate on the data as appropriately. Note that the flag is reset at the time the data is used.

### Data Protocol

The Home Assistant/MySensors protocol works pretty well for binary data. Not well understood is why there are so many names for true and false: Sometimes it’s a switch, and sometimes light or something else. Sending numeric data from MySensors to Home Assistant is straightforward, (one should read the [MySensors Serial Protocol 2.x](https://www.mysensors.org/download/serial_api_20))

Sending data from Home Assistant to MySensors is arcane. When setting up the MyMessage the type must be V\_TEXT. When presenting, the type is S\_INFO. And, as noted, MySensors must send a value to Home Assistant. Now here's the trick. In Home Assistant, even though the type is *text* if the text is also a number, MySensors can and will read it as a number. eg. message.getLong();

# Some things that should be noted

## [Reading the battery voltage](https://forum.arduino.cc/t/arduino-secret-voltmeter-explanation/447711).

## Device and Sensor Nomenclature

When only one program is going to be maintained and there are multiple devices using the code, there must be a way to distinguish between the various devices. In MySensors, each device is assigned a node id and that is stored in the EEPROM at address 0, allowing for 254 devices (0 and 255 are reserved). If the value found in the EEPROM is 255, MySensors will assign an arbitrary value. If an arbitrary number is not convenient, which it usually isn’t, employ a separate program to set the 0 address of the EEPROM.

When the sketch name is presented to Home Assistant, Home Assistant will know the device as the sketch name with the node ID appended. But the name of a sensor is exactly the same name that another device uses, Home assistant appends an incrementing number to the sensor name. This is inconvenient when trying to identify a specific sensor on a specific device.

To avoid this inconvenience, when a sensor is presented, the node id is appended to the sensor name. Unfortunately, it appears that sensor names can only have six characters, which means the base sensor name can only have three characters.

In this code a *string* (not *String*) is created that ends with the node ID (*prepresentation()*). In *presentation()* the sensor name is overwritten in the first three characters of the *string*.

# Schematic

In lieu of a schematic

**Connector sensor Arduino TP4056 Solar Panel**

J12 VCC 3V3

J13 GND GND

J11 AOUT A0

J10 J3/J4 GND OUT-

J9 J3/J4 5V OUT+

J1/J2 + red wire

J1/J2 - black wire

J14 TX (optional)

J15 RX (optional)

J5/J6 extension cable for solar panel (optional)

J7/J8 (not used)

1. An [Arduino Nano](https://www.aliexpress.us/item/3256806880365712.html) and [nRF24 radio](https://www.aliexpress.us/item/3256807580580500.html) can be used ([ref](https://www.mysensors.org/build/connect_radio)) [↑](#footnote-ref-0)
2. wait() allows MySensors background process to run while delay() does not. [↑](#footnote-ref-1)