Low-cost LoRa Collar for cattle rustling applications

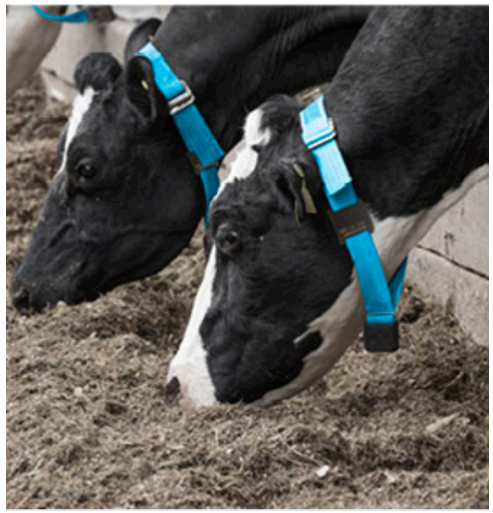
Prof. Congduc Pham
http://www.univ-pau.fr/~cpham
Université de Pau, France

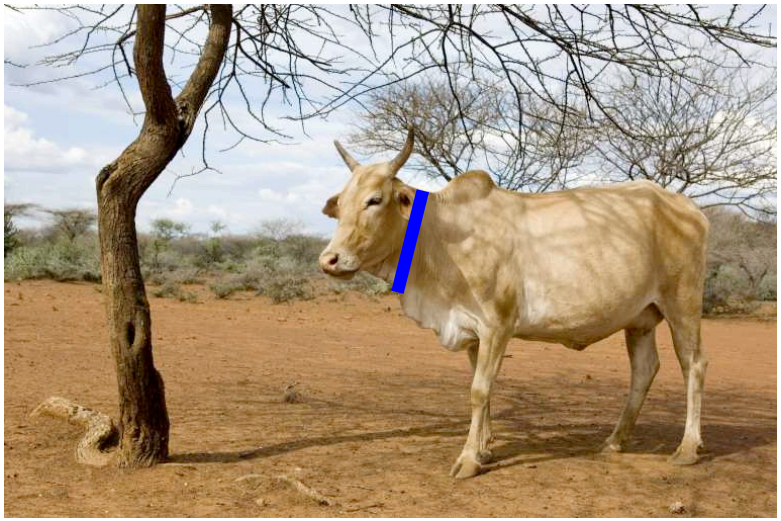
Overview

- We will show how to build a simple low-cost collar for preventing cattle rustling in developing countries. The system is proposed in 2 versions
- **Version 1: simple beacon**
  - When powered on, sends every 10 minutes a beacon
  - The distance can be estimated with the beacon's RSSI
  - The gateway will receive the beacon messages and tries to detect whether an alarm should be raised or not (no beacons for instance or very low RSSI)
- **Version 2: GPS beacon with GPS module**
  - When powered on, sends every 10 minutes a GPS beacon
  - The position can be determined with the GPS coordinates
  - The gateway will receive the GPS beacon messages and tries to detect whether an alarm should be raised or not (no beacons for instance or out-of-range position)

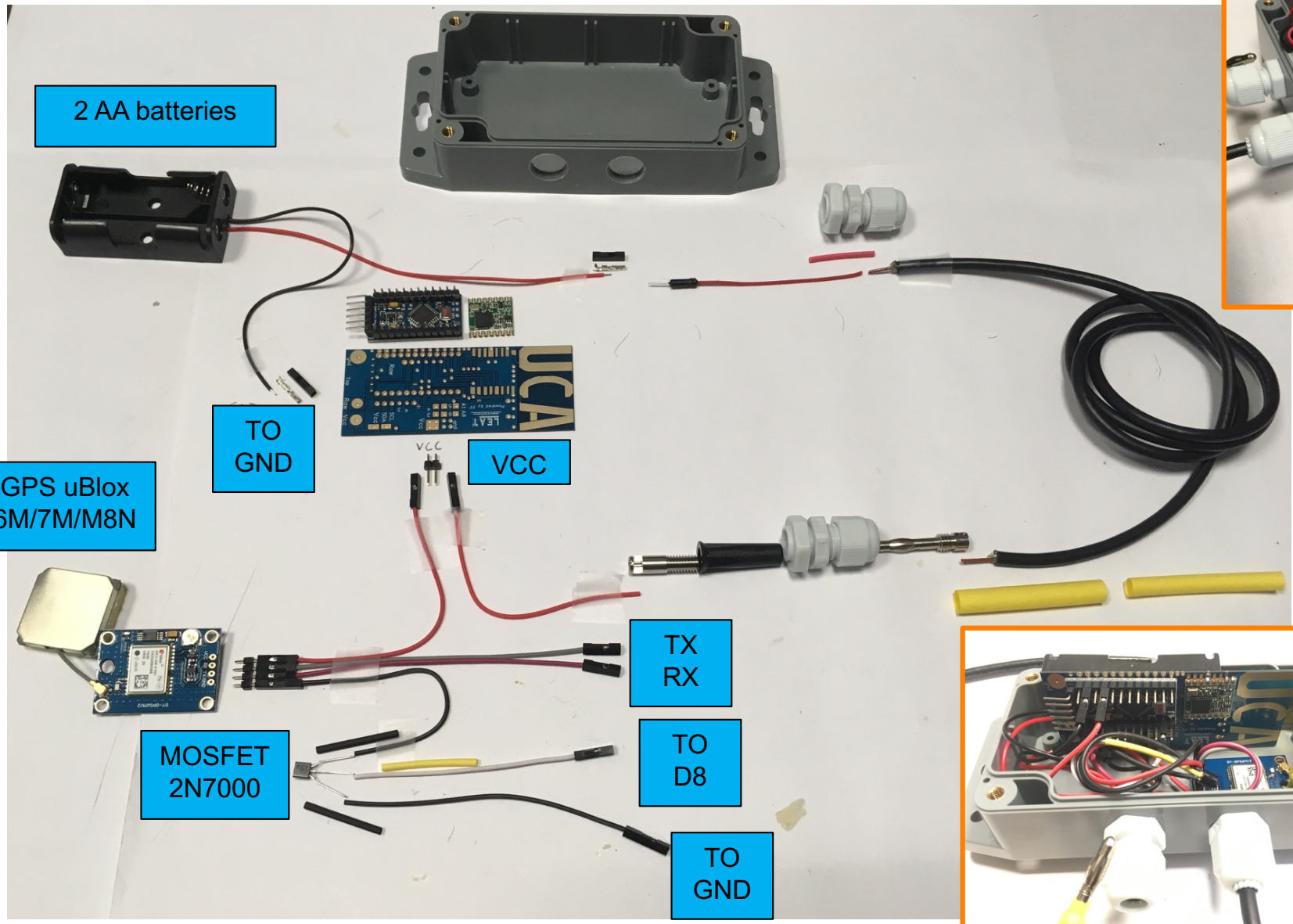
How it works?

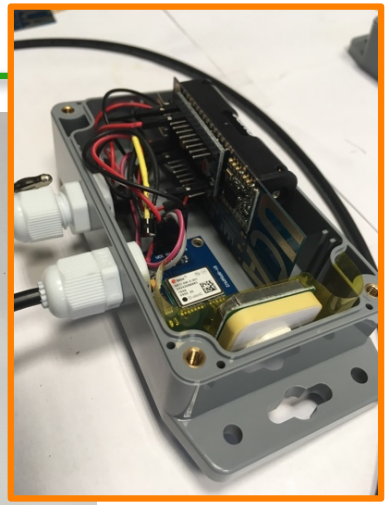
- The collar will be fixed to the cow, around neck. Example picture from Afimilk Silent Herdsman for health monitoring
- In our case, reception of beacon means that the cattle is in range (with GPS, the exact position can be determined)
- If out-of-range, disconnected or damaged device, an alarm can be raised
- To detect collar cutting, the power wire will also goes around the catlle's neck





SUMMARY OF PART LIST





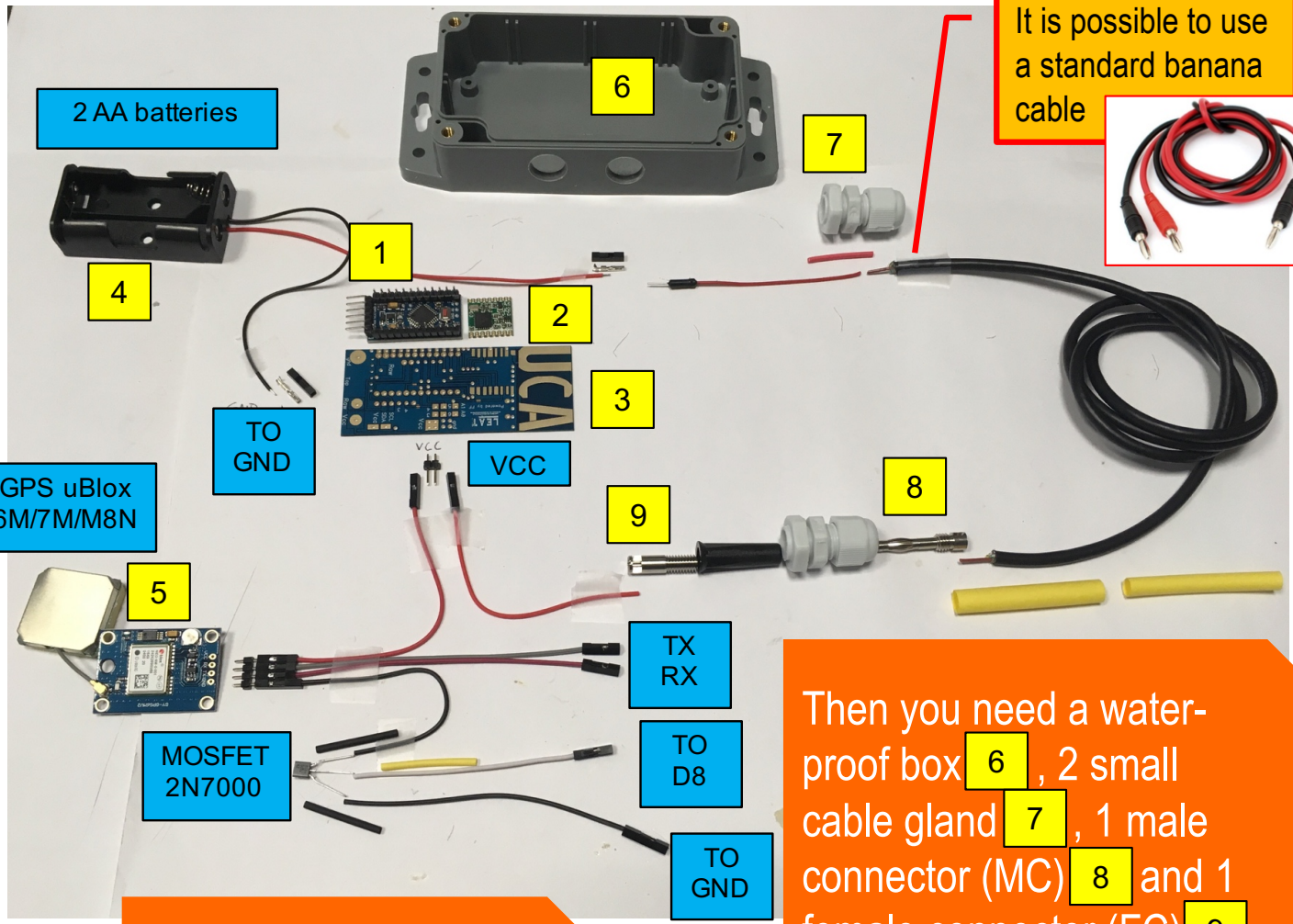
Description

2 AA batteries 4 will power the board with an autonomy of several months

An optional GPS module can be added 5

Use an Arduino Pro Mini 3.3v at 8MHz 1

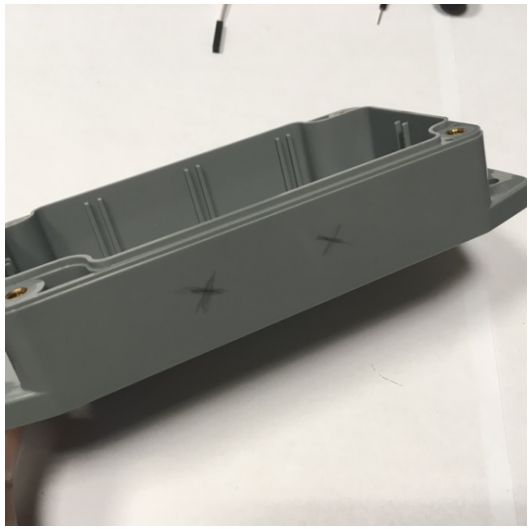
Radio module is an RFM95W 2

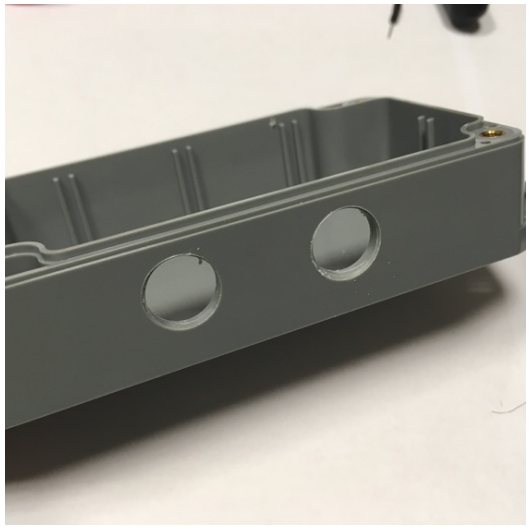


A PCB with integrated antenna will be used 3

Then you need a water-proof box 6, 2 small cable gland 7, 1 male connector (MC) 8 and 1 female connector (FC) 9

Drill holes for connectors





Drill 2 holes depending on the gland diameter (here 12mm, so hole of 13mm)

Place the cable glands

Use other type of connectors

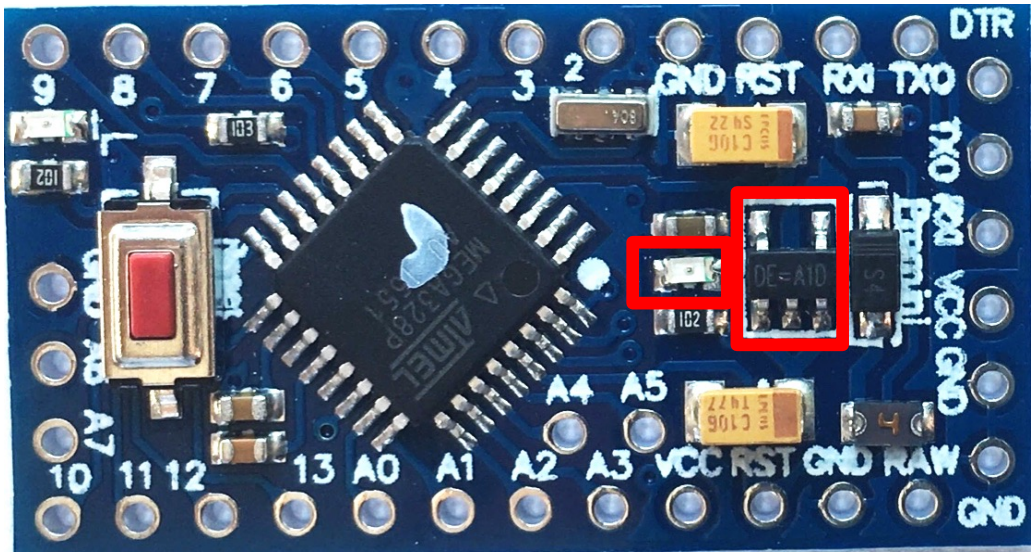
Using female banana connector can be a better solution as you can use a standard banana cable

An even better solution can use aviator-type waterproof plugs

You can adapt/mix at your convenience!

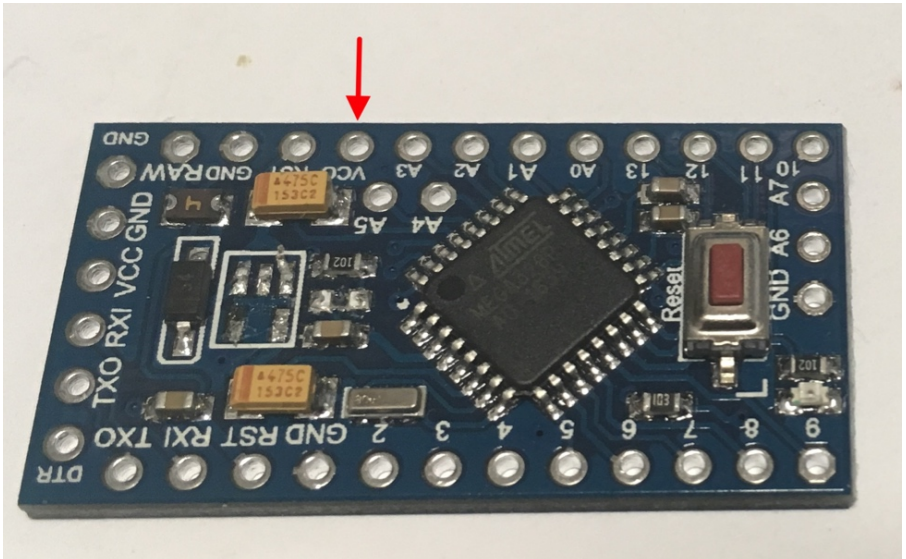
Extreme low power (1)

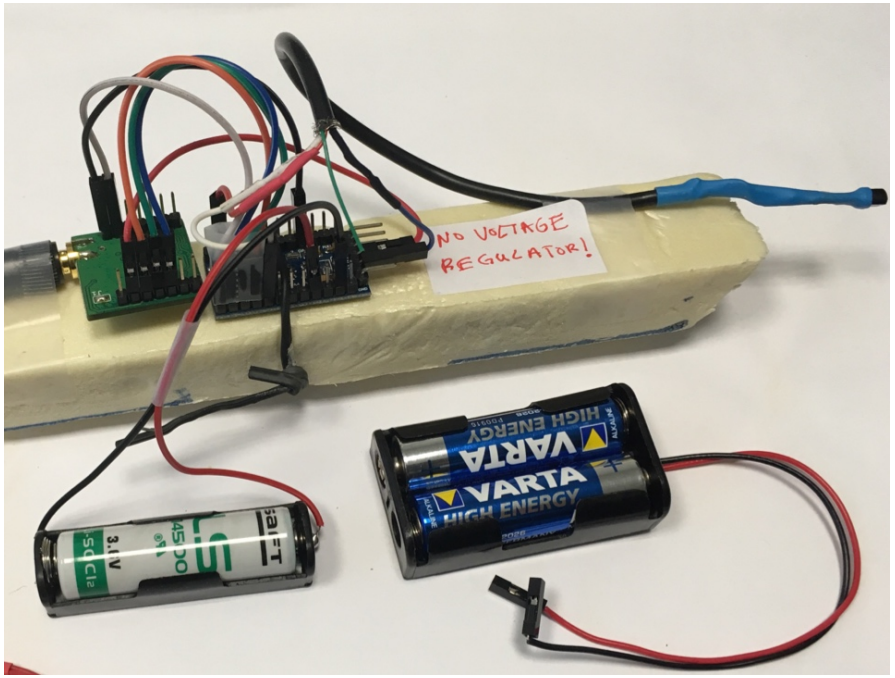
- You can greatly reduce the power consumption by removing the power led (left box) and the voltage regulator (right box): 5µA in sleep mode
- See our "Extreme low-cost & low-power LoRa IoT DIY" video tutorial at https://www.youtube.com/watch?v=2_VQpcCwdd8



Extreme low power (2)

- Then, inject directly between 3.2v and 3.6v into the VCC pin instead of the RAW pin
- You can use 2 AA batteries (3.2v) or a 3.6v Lithium battery

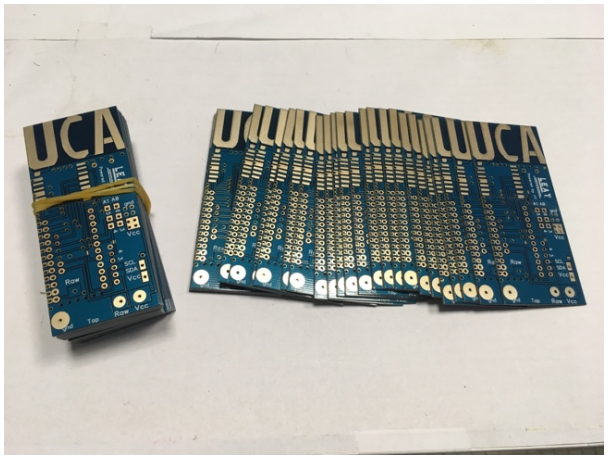


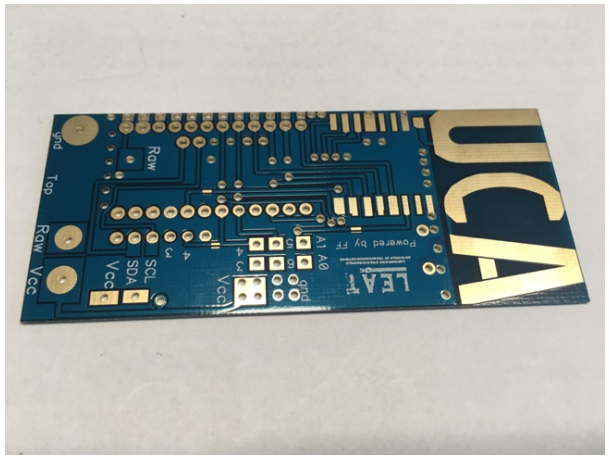


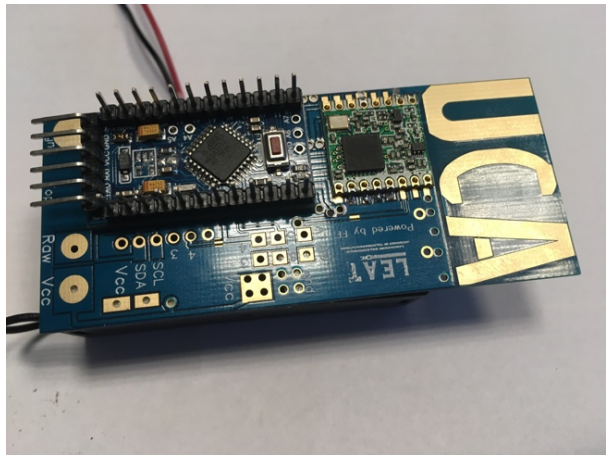
WARNING, do not inject more than 3.6v when the regulator is removed! You can destroy your board!

PCB WITH INTEGRATED ANTENNA

- A PCB with an integrated antenna can be used
  - To facilitate the integration of the Arduino Pro Mini and a smaller LoRa radio module (HopeRF RFM95W)
  - To avoid having an external antenna that can be very fragile



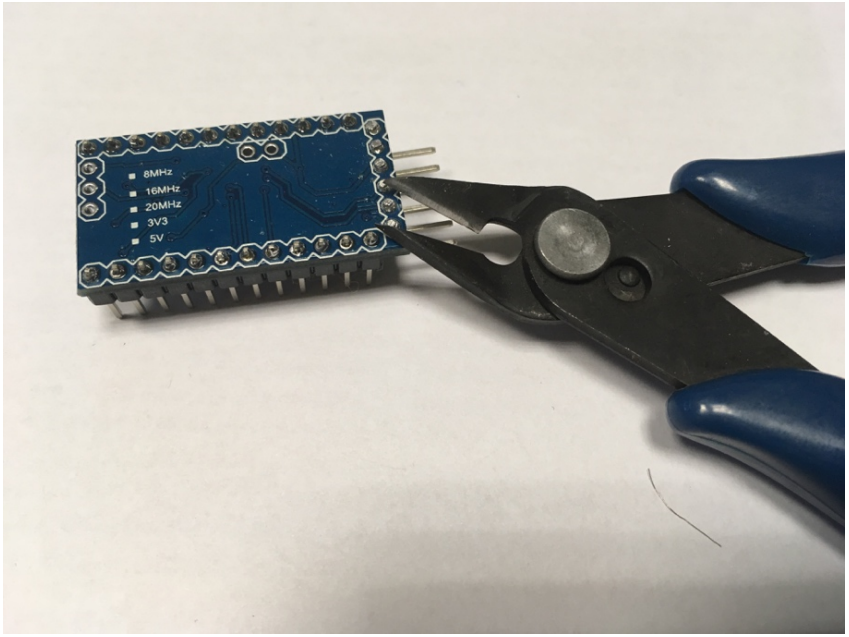




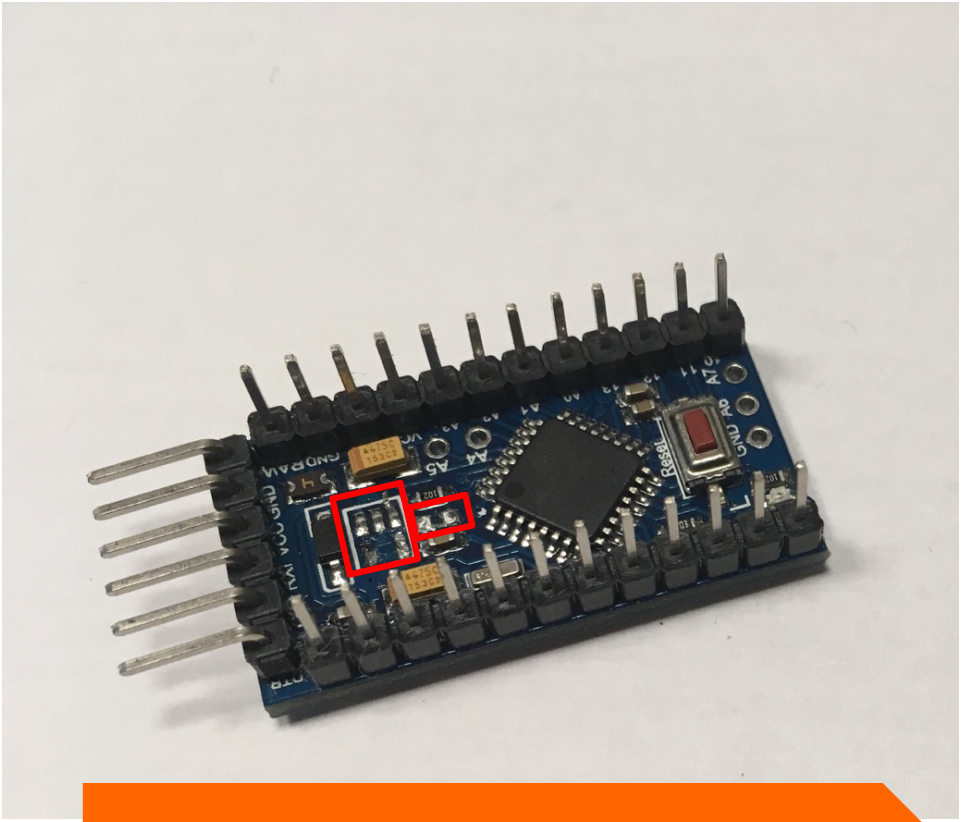
PCB w/antenna designed by Fabien Ferrero from LEAT, University Côte d'Azur, France
The PCB Gerber layout file is available and making can be ordered from Chinese manufacturers

https://github.com/FabienFerrero/UCA_Board

Preparing the Pro Mini

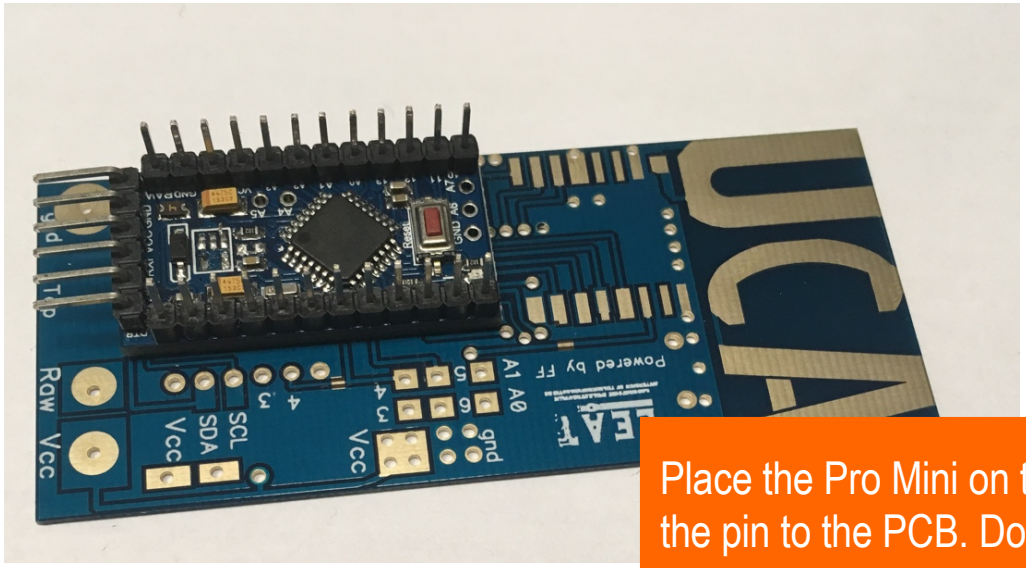


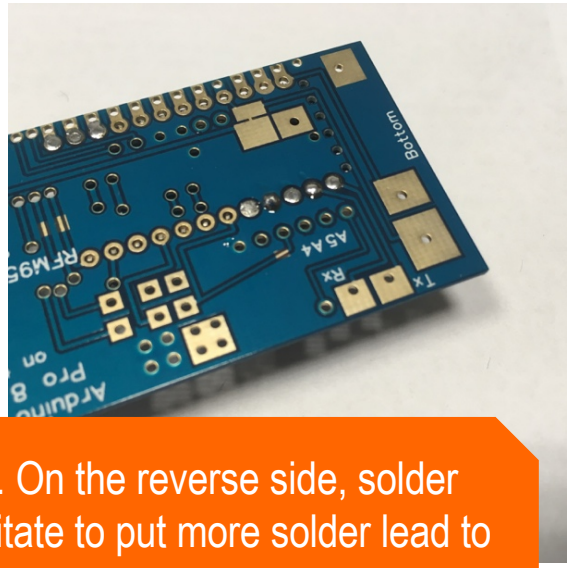
Solder the header pin. Be careful to not put too much solder lead because we need to further solder the board to the PCB. Cut the extra length on the reverse side of the 6-pin programming header so that it will not touch the PCB.

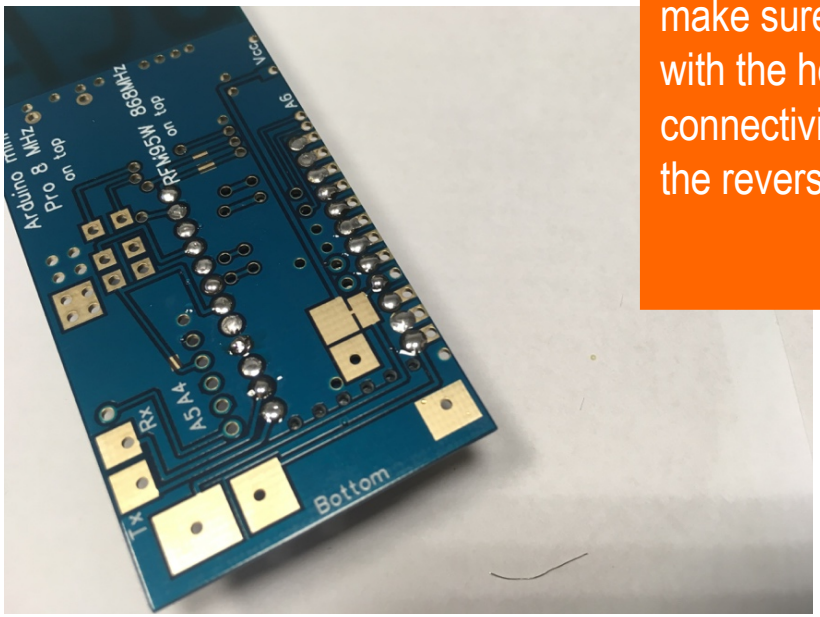


Remove the power led with the tip of a cutter and the voltage regulator with a small plier.

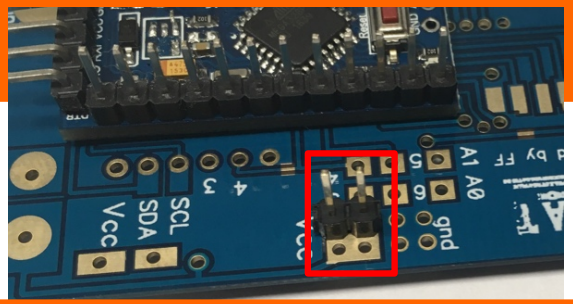
Soldering to the PCB





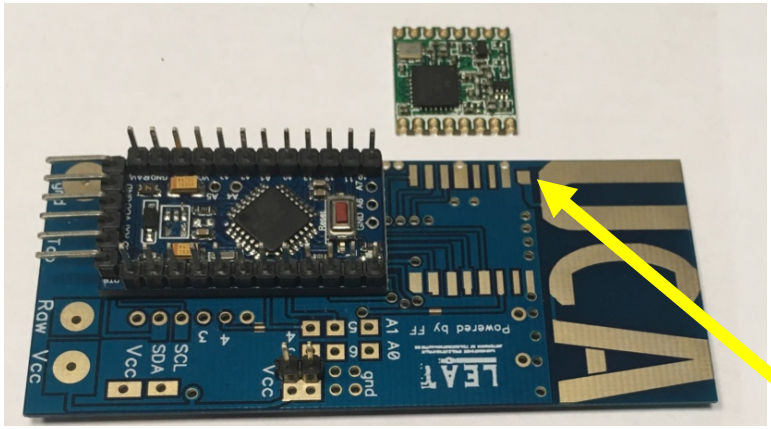


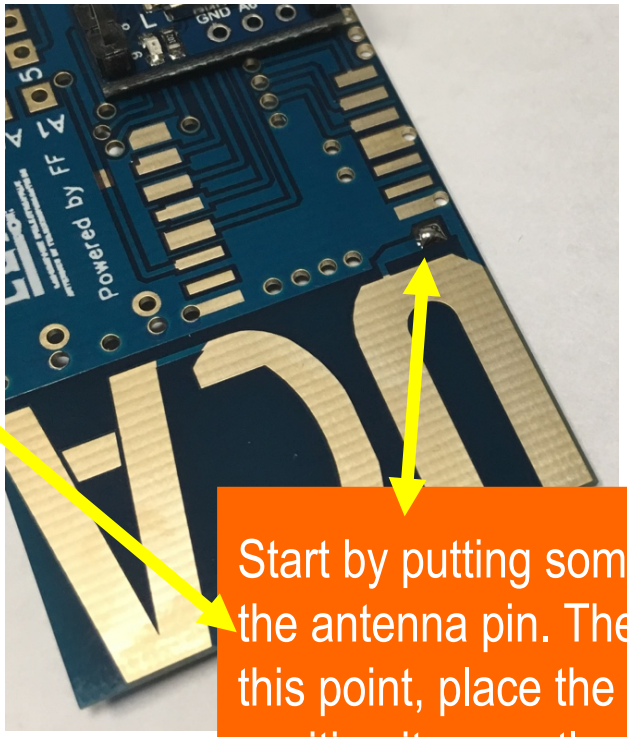
Place the Pro Mini on the PCB. On the reverse side, solder the pin to the PCB. Do not hesitate to put more solder lead to make sure that it goes through the hole and make contact with the header pin. When done, it is recommended to check connectivity (with a multimeter) between each solder point at the reverse side and the corresponding pin on the Pro Mini.

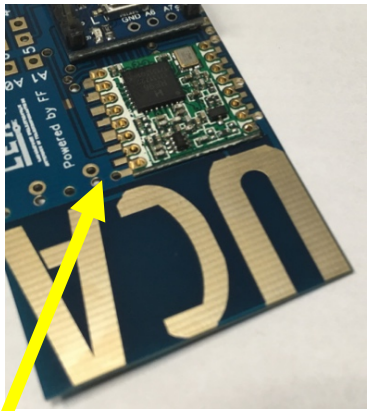


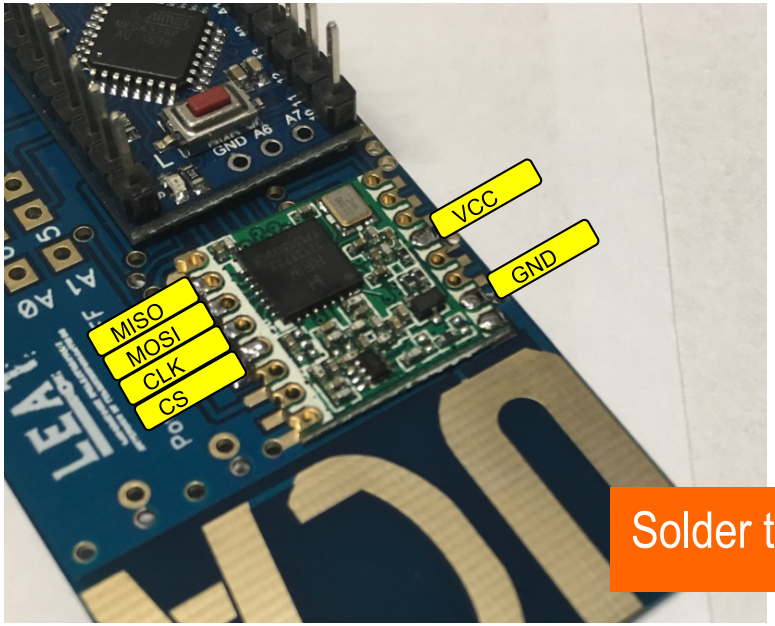
Then solder a 2-pin header for VCC.

PUT THE RADIO MODULE





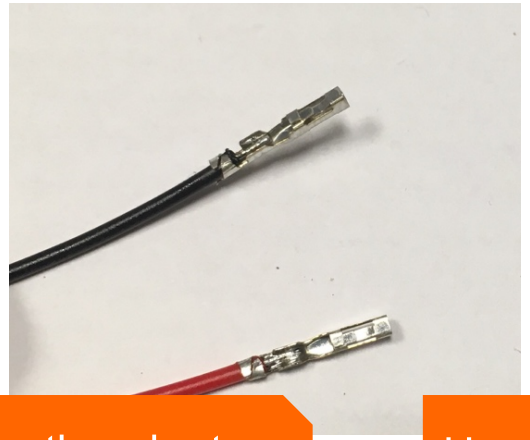


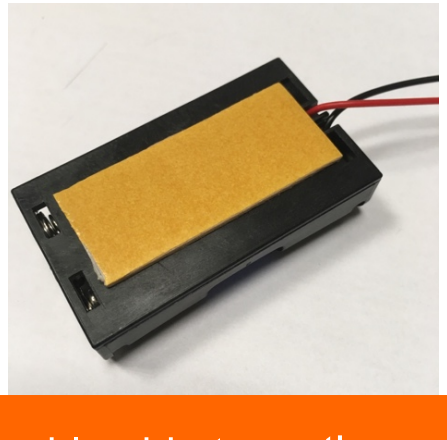


Start by putting some solder lead on the antenna pin. Then, while heating this point, place the radio module and position it correctly.

Solder the other pins of the radio module

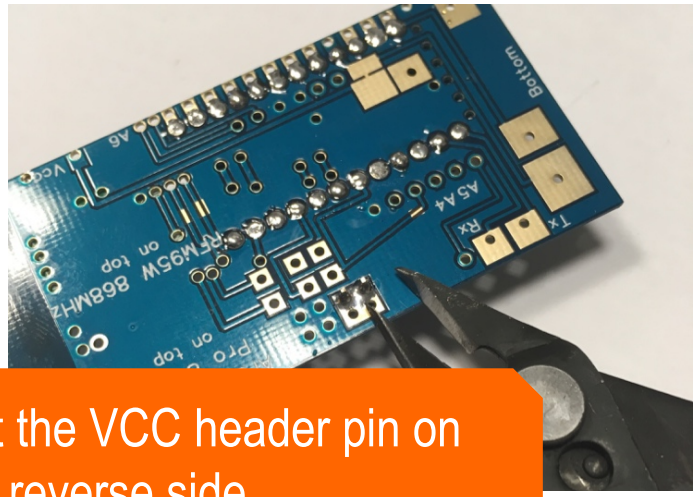
The battery pack

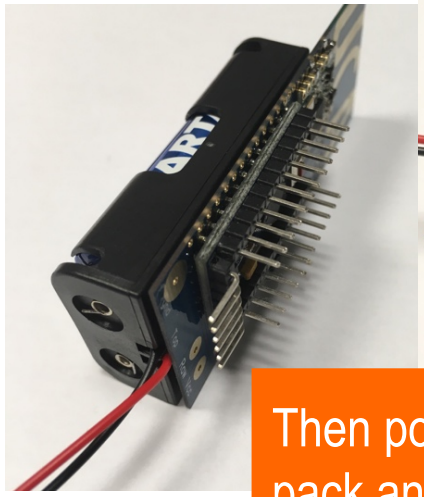


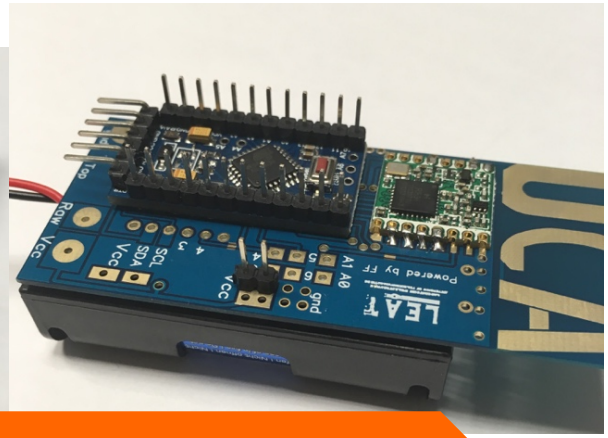


It is recommended to solder the wire to the Dupond connector

Use double-side tape, those used to fix mirror on walls





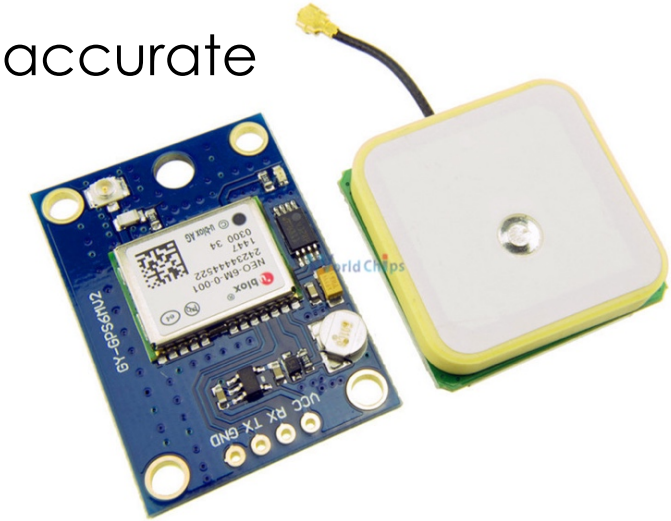


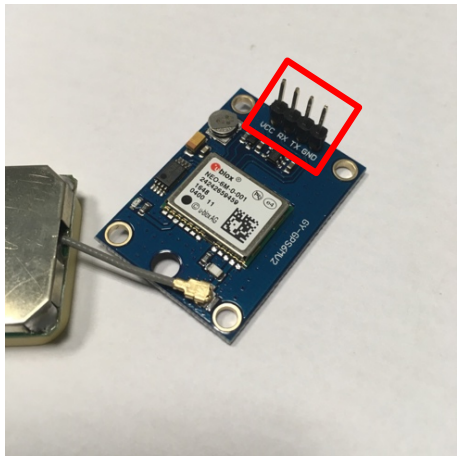
Cut the VCC header pin on the reverse side

Then position the battery pack and press firmly

Adding a GPS module

- Here, a GPS module is added to get an accurate positioning system
  - We use ublox-based GPS modules
  - These modules can be easily obtained from Chinese integrators
  - Ublox NEO-6M are sufficient and their cost is lower: about 6€.
  - Ublox NEO 7M/M8N are supported and have lower power consumption (55mA instead of 75mA)





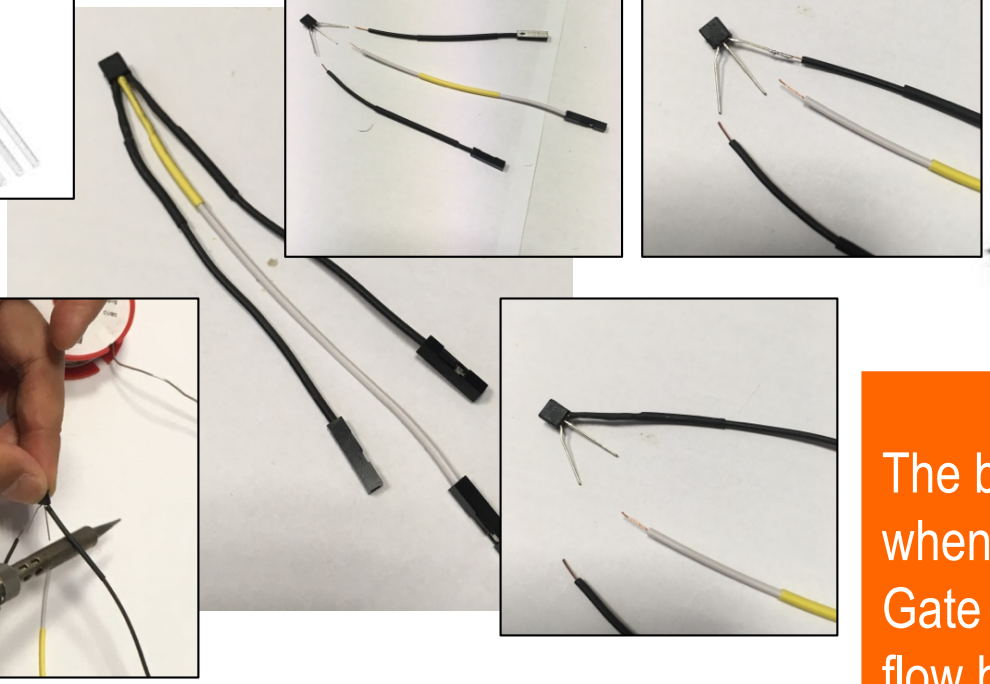
Solder a 4-pin header if needed

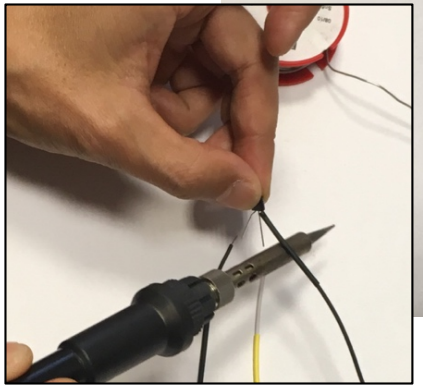
Low power for GPS

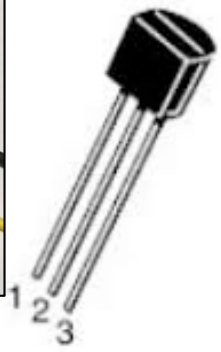
- The best option to reduce the GPS power consumption is to completly switch off the GPS between 2 wake up
- The first GPS fix is usually obtained in about 30s
- Each time the GPS is powered on again it can be considered as a cold start but a GPS fix can usually be obtained in about 4s to 7s
- We will use digital pin 8 coupled to a MOSFET transistor to realize a very low cost and simple ON/OFF switch

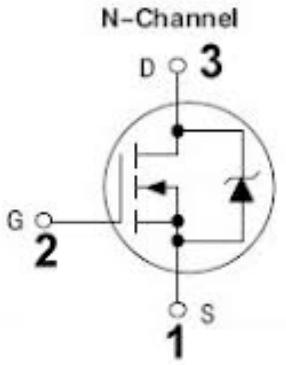
Using the 2N7000 MOSFET

See the heat-shrinking sleeves









We use a 2N7000 MOSFET transistor where the maximum current of 200mA is sufficient to power the GPS (about 60mA in acquisition mode)

The basic principle is as follows: when the voltage V_{GS} applied on the Gate pin (G) is sufficient, current can flow between the Drain (D) and the Source (S). With the 2N7000, V_{GS} is typically 2v with a max of about 3v but it can work with the output of the Arduino Pro Mini 3.3v digital pin

CONNECTING THE 2N7000

GPS GND

N-Channel

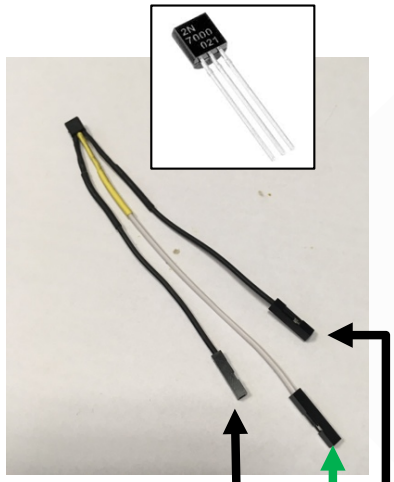
D 3

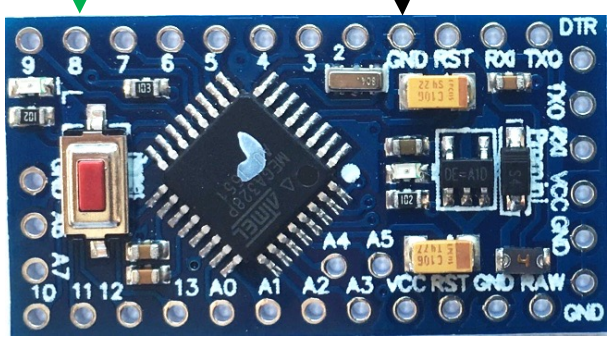
µC pin 8

G 2

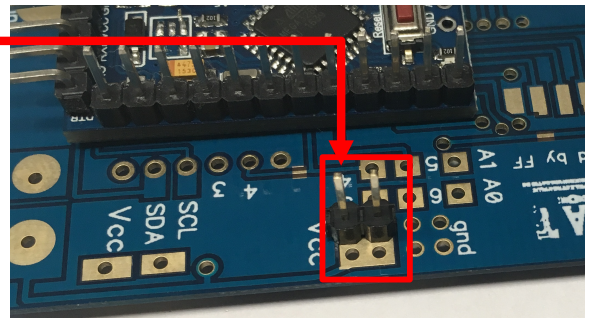
S 1

µC GND





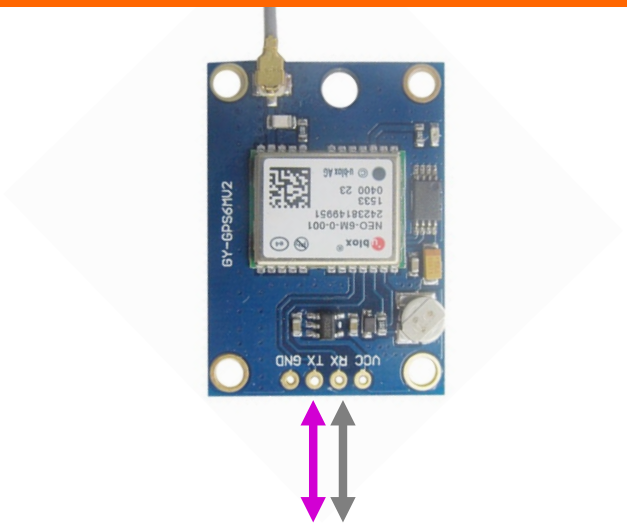
In acquisition mode, the device consumes about 55mA (NEO 7M/M8N) and about 75mA (NEO 6M). In sleep mode with GPS off, it consumes only 5µA!

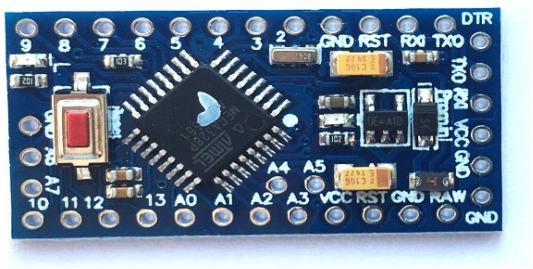


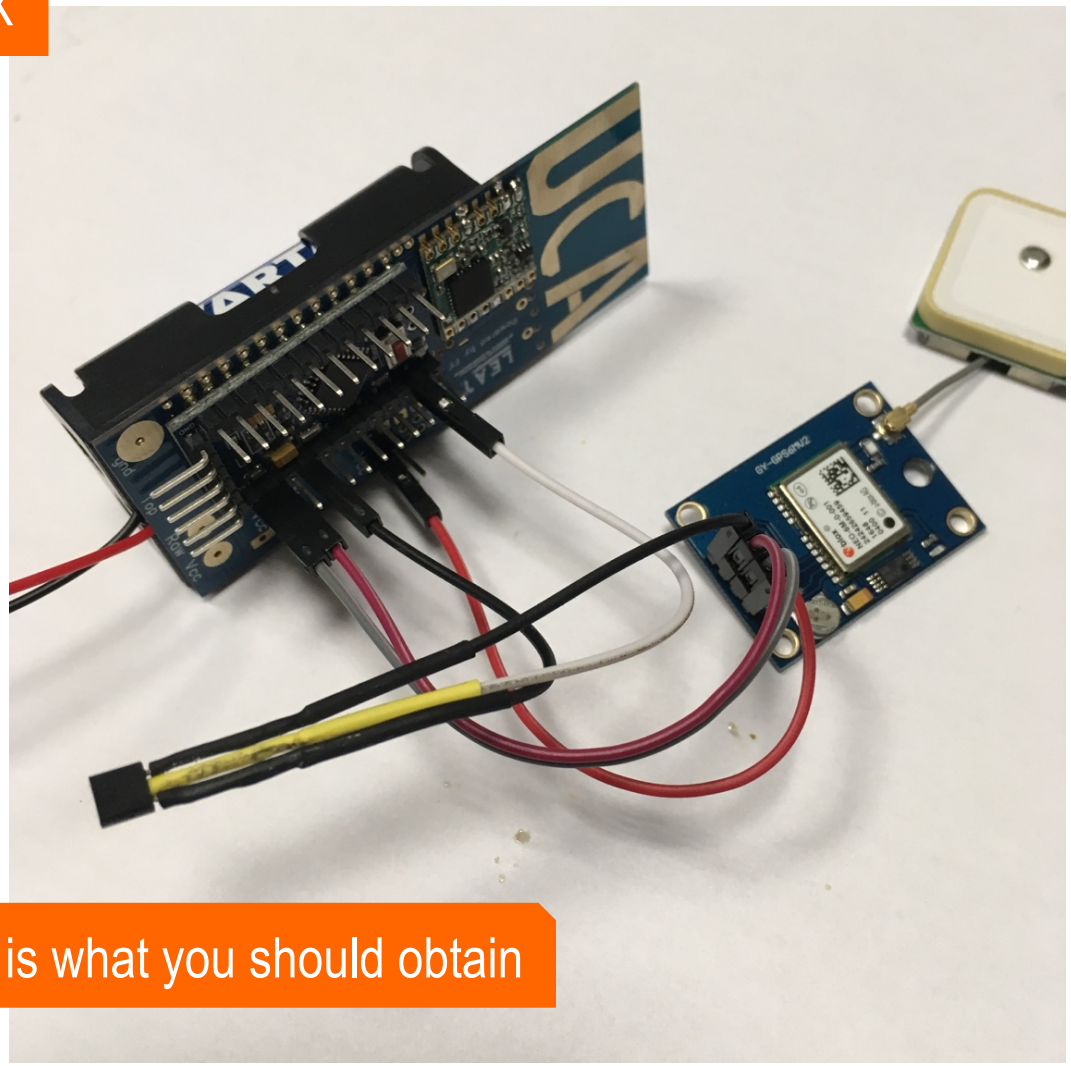
The GPS VCC is directly connected to one of the PCB's VCC that will deliver 3.3v

Connecting the GPS

GPS uses serial, so connect TX and RX

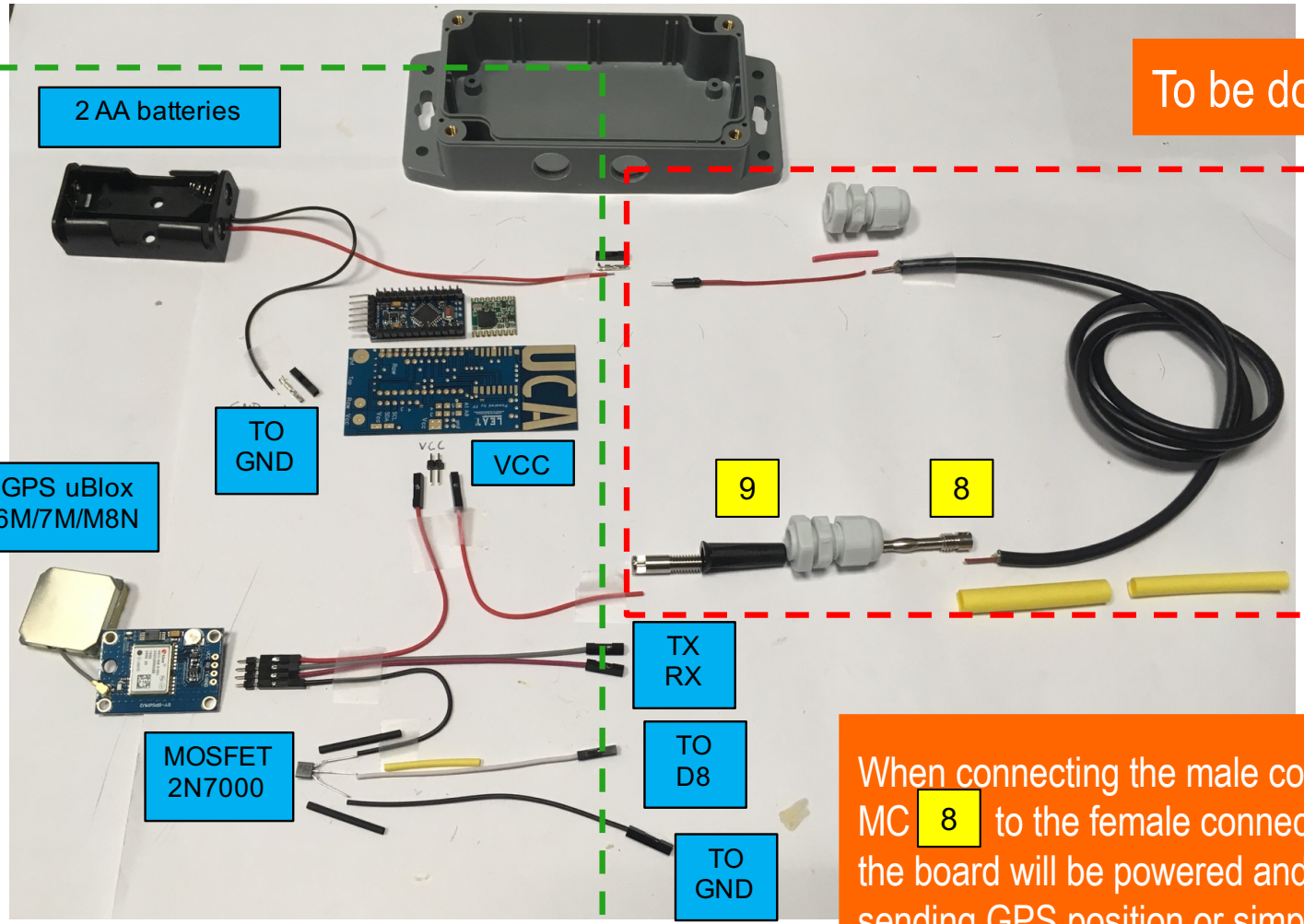




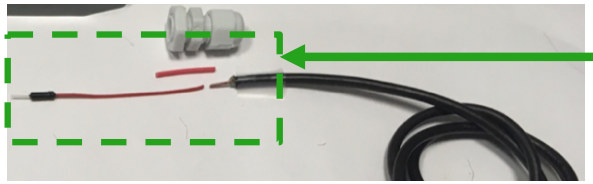


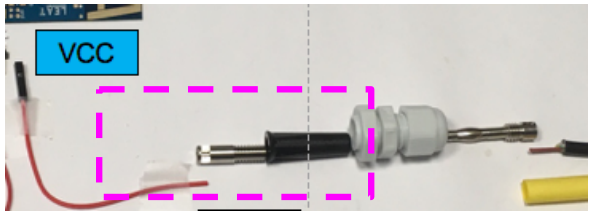
This is what you should obtain

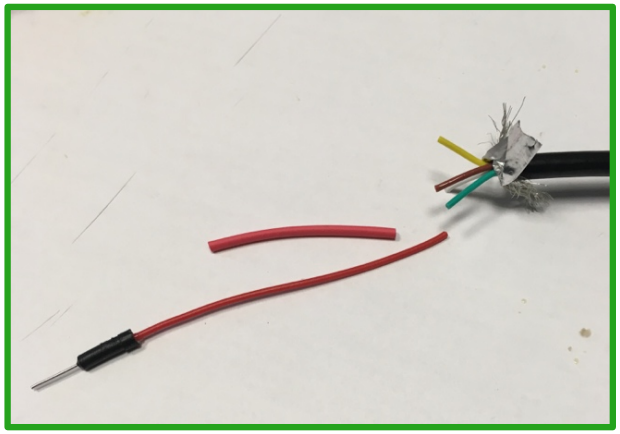
The powering system

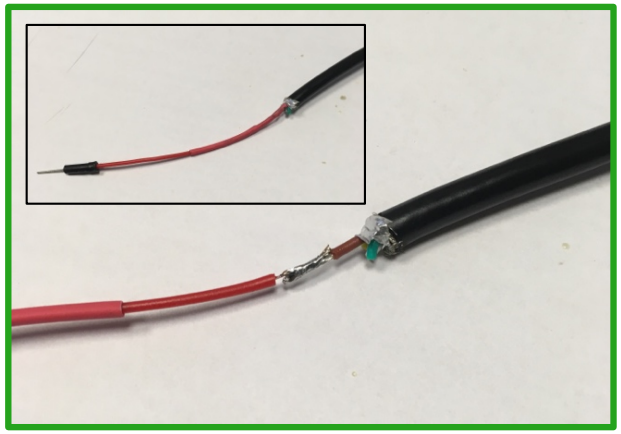


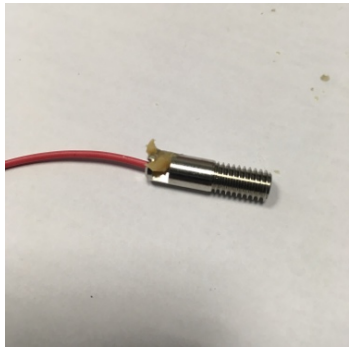
100% DIY CABLE (1)

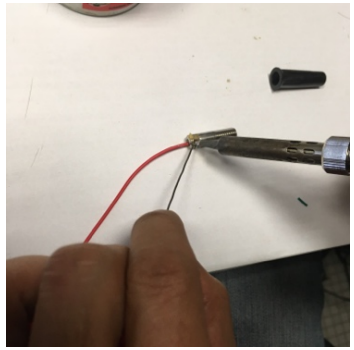




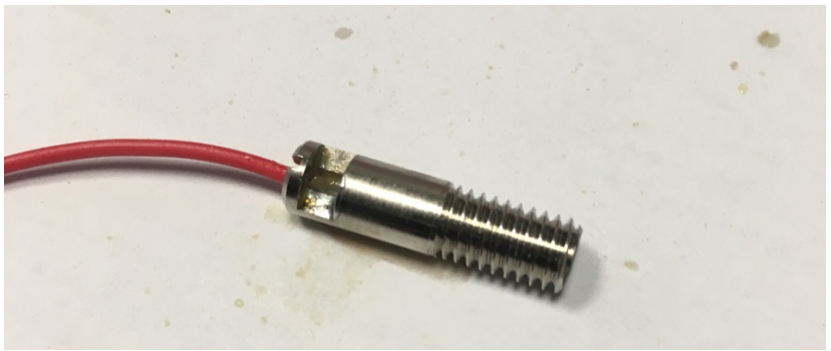




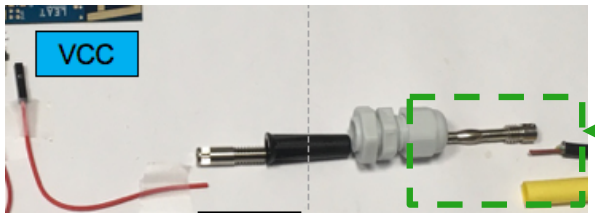




Use some solder paste to have better result when soldering the wire to the connector

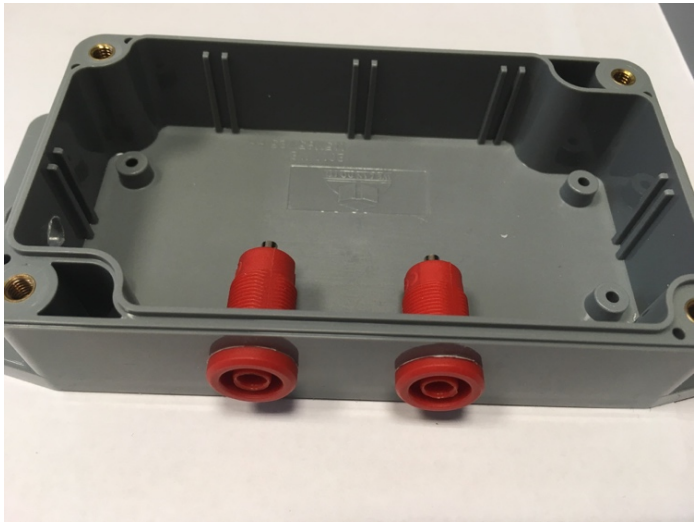


100% DIY CABLE (2)

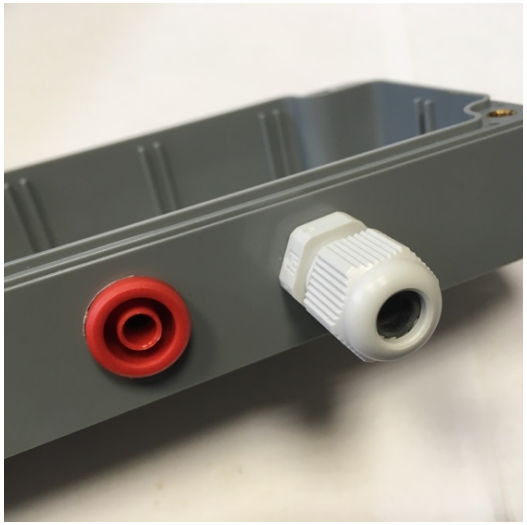


Notice the 2 heat-shrinking sleeves. The first one has larger diameter.

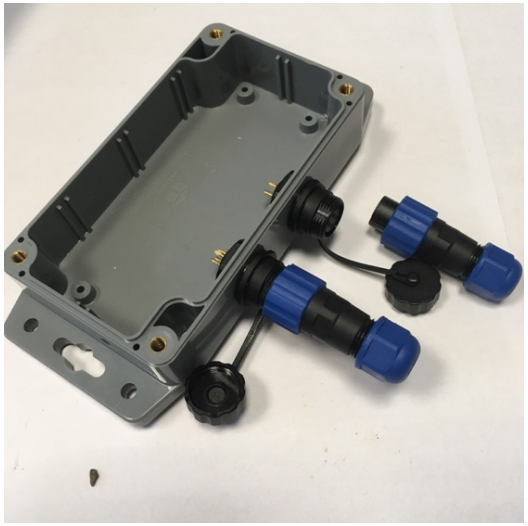
Mixed approach



Using female banana connector can be a better solution as you can use a standard banana cable



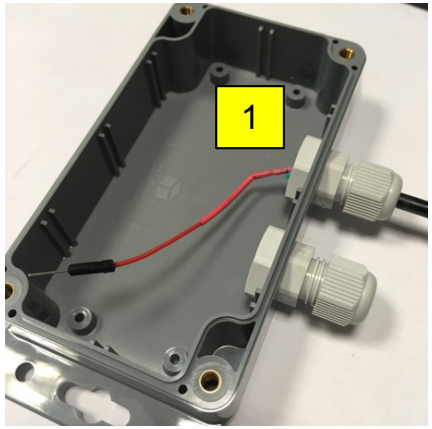
If you need a custom length, you can cut one end of the banana cable

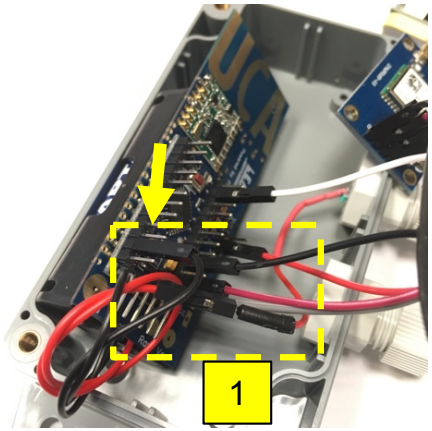


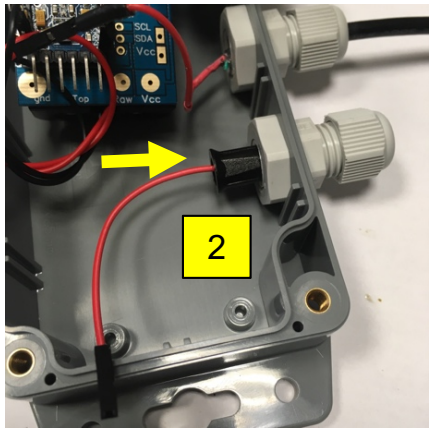
An even better solution can use aviator-type waterproof plugs

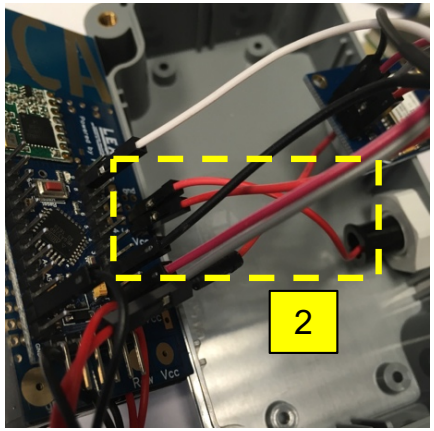
You can adapt/mix at your convenience!

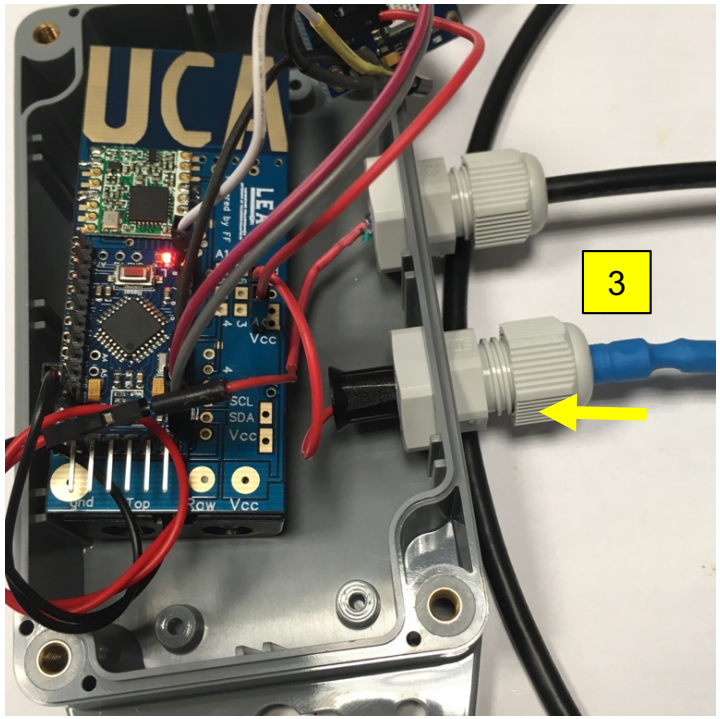
PUT HARDWARE IN PLACE









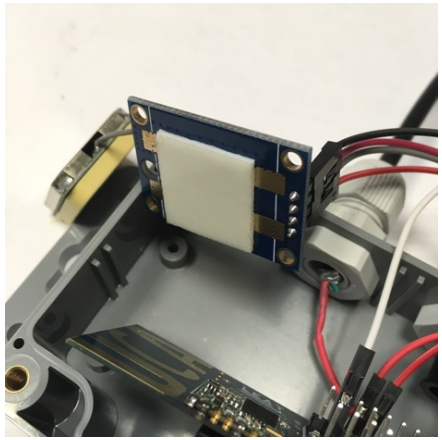
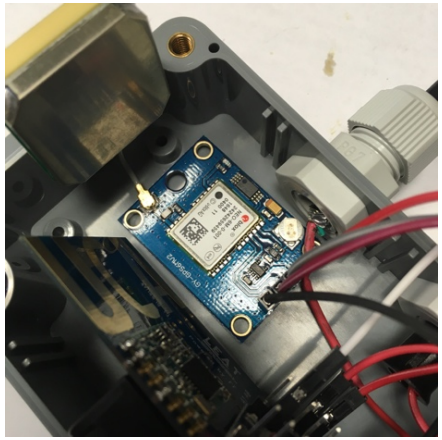
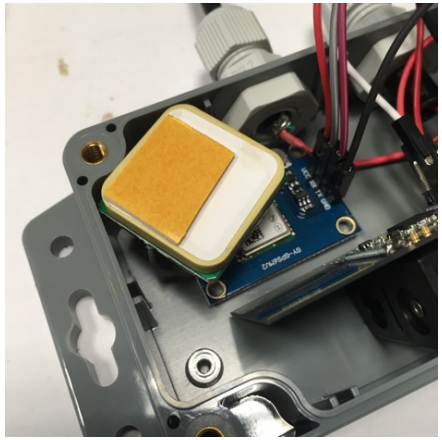
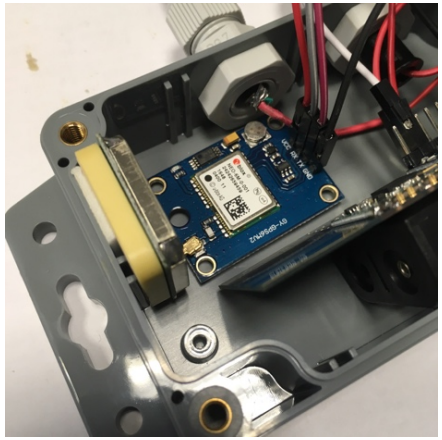


Connect the VCC from the battery pack to the first end of the cable 1, through the cable gland. Connect GND from battery pack to GND on the Pro Mini.

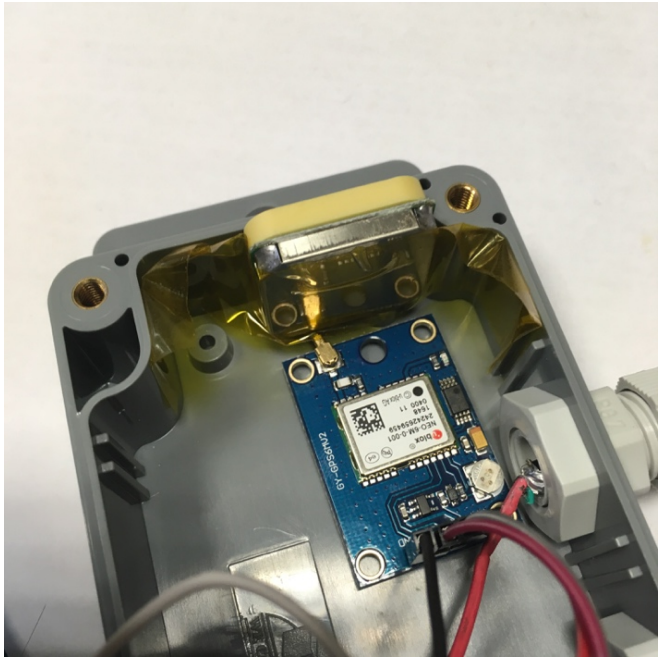
Insert strongly the female connector in the cable gland, put some glue if necessary 2. Then connect the wire to the other VCC pin on the PCB.

When plugging the male connector to the female connector 3, the system will be powered.

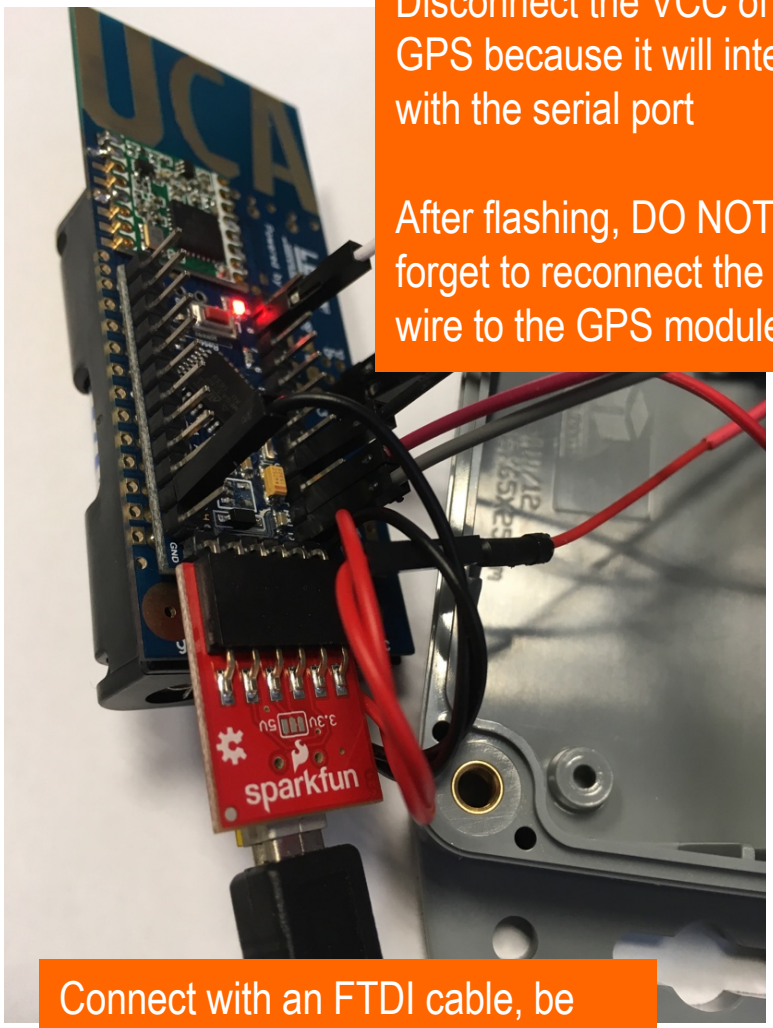
Fixing the GPS module

Again use double-side tape, those used to fix mirror on walls, to fix the GPS module to the box. Do the same for the antenna. Add tape if necessary, to secure the GPS antenna.



Flashing the device

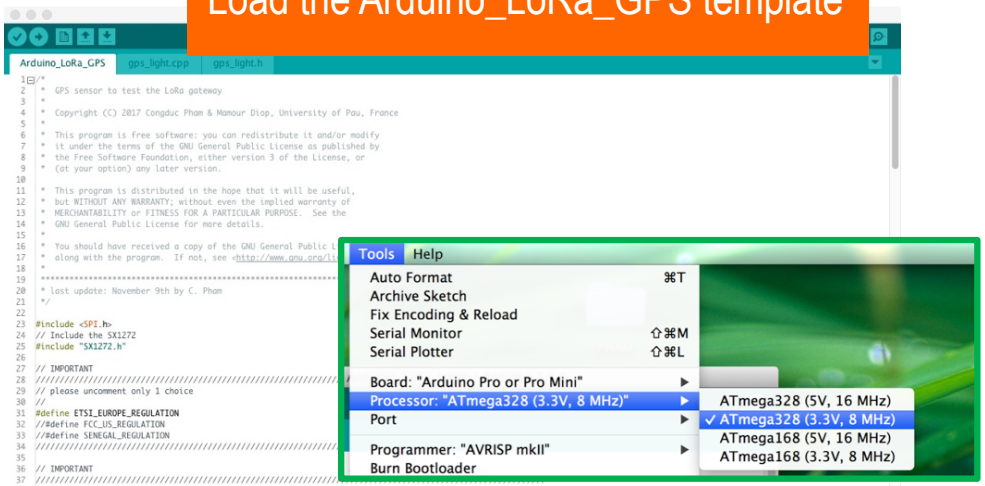


Disconnect the VCC of the GPS because it will interfere with the serial port

After flashing, DO NOT forget to reconnect the VCC wire to the GPS module

Connect with an FTDI cable, be sure to match the VCC pin to the one of the programming header

Load the Arduino_LoRa_GPS template

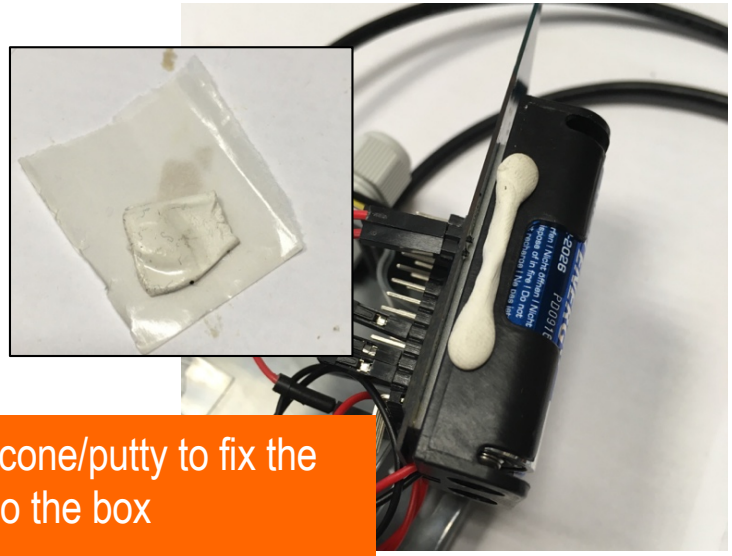


Select Arduino Pro Mini, 3.3v and 8MHz and select the right communication port

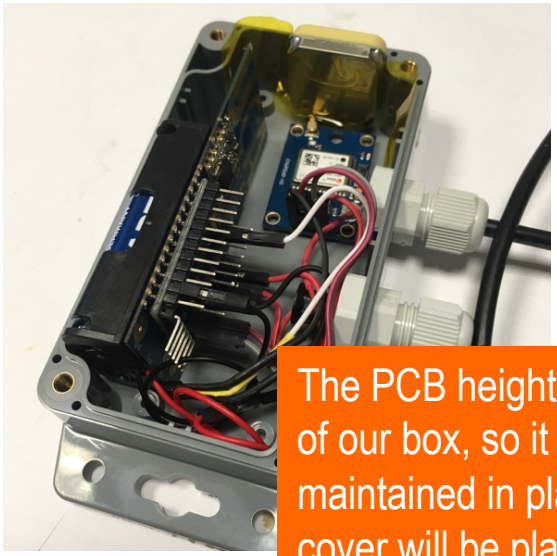
```
108
109  ////////////////////////////////////////////////////////////////////////
110  // CHANGE HERE THE TIME IN MINUTES BETWEEN 2 READING & TRANSMISSION
111  unsigned int idlePeriodInMin = 20;
112  ////////////////////////////////////////////////////////////////////////
113
114  #define GPS_FIX_ATTEMPT_TIME_IN_MS 35000
115
```

Change the wake-up (idlePeriodInMin) time if needed

FIXING THE BATTERY PACK

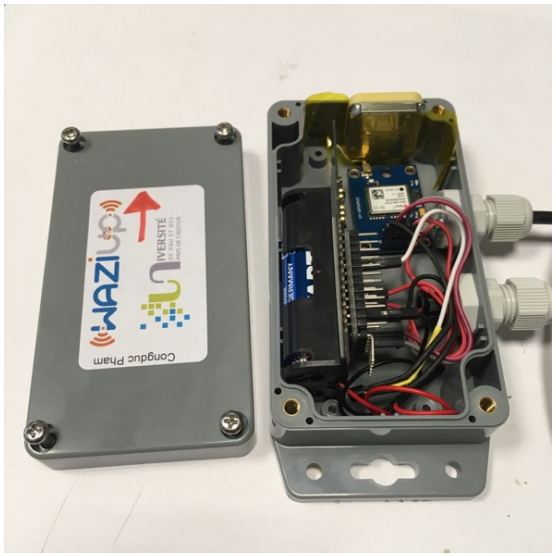


Use some silicone/putty to fix the battery pack to the box



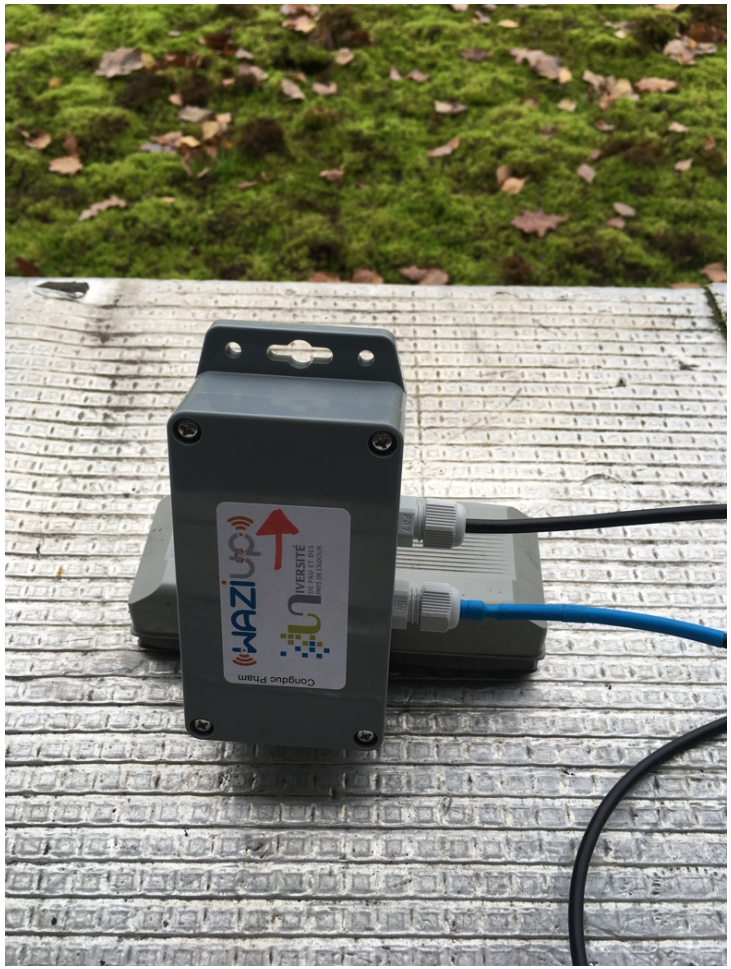
The PCB height is exactly the one of our box, so it will also be firmly maintained in place when the box cover will be placed

Mark the UP position



The final result!

Ready for testing



The default configuration in the Arduino_LoRa_GPS example is:

Use LoRa mode 1. Send GPS to the gateway every 20 minutes
Node short address is 15. Digital pin 8 drives the MOSFET.

BC (beacon counter) starts at 0, increases by 1 at each beacon, returns to 0 after 65536 beacons

A GPS fix will be attempted during 35s maximum. After that time, a beacon with be sent. If the fix is unsuccessful, then the message is: \!BC/0/LAT/0/LGT/0/FXT/-1

If the fix is successful the message is:

\!BC/0/LAT/43.31448/LGT/-0.36491/FXT/26271

Where FXT is the time to get the fix in ms

STARTING THE GATEWAY

We just use the simple low-level gateway here

```
pi@raspberrypi:~/lora_gateway $ sudo ./lora_gateway
SX1276 detected, starting.
SX1276 LF/HF calibration
...
^$**********Power ON: state 0
^$Default sync word: 0x12
^$LoRa mode 1
^$Setting mode: state 0
^$Channel CH_10_868: state 0
^$Set LoRa power dBm to 14
^$Power: state 0
^$Get Preamble Length: state 0
^$Preamble Length: 8
^$LoRa addr 1: state 0
^$SX1272/76 configured as LR-BS. Waiting RF input for transparent RF-serial bridge
^$Low-level gw status ON
--- rxlora. dst=1 type=0x12 src=15 seq=0 len=29 SNR=8 RSSIpkt=-43 BW=125 CR=4/5 SF=12
^p1,18,15,0,29,8,-43
^r125,5,12
^t2017-12-19T12:55:57.473
?\!BC/0/LAT/0/LGT/0/FXT/-1
```

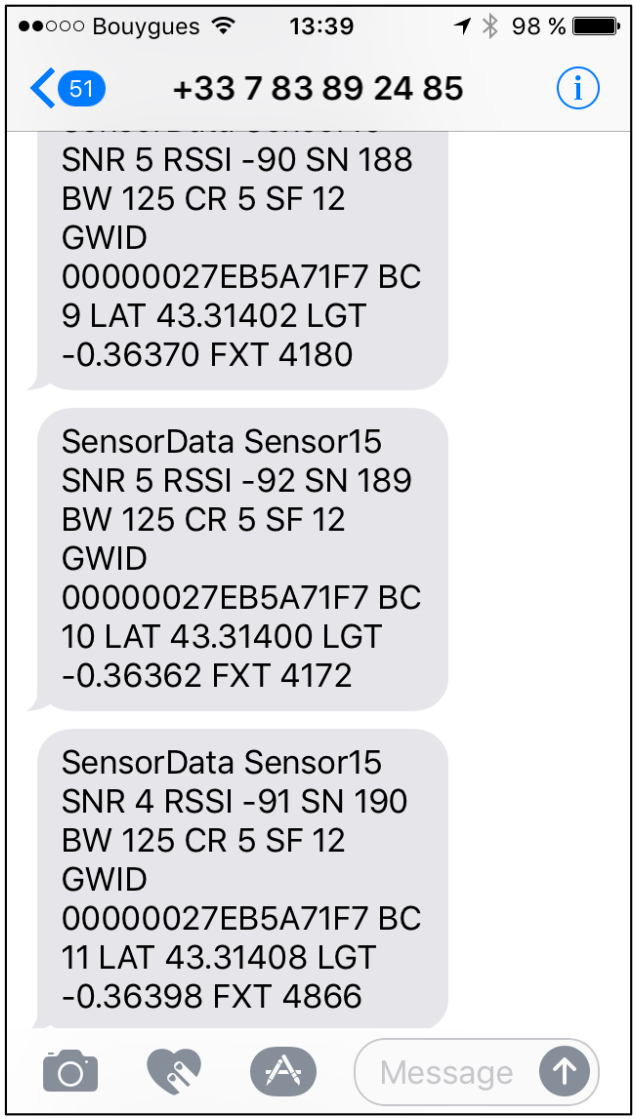
The test is done indoor and the fix was not successful

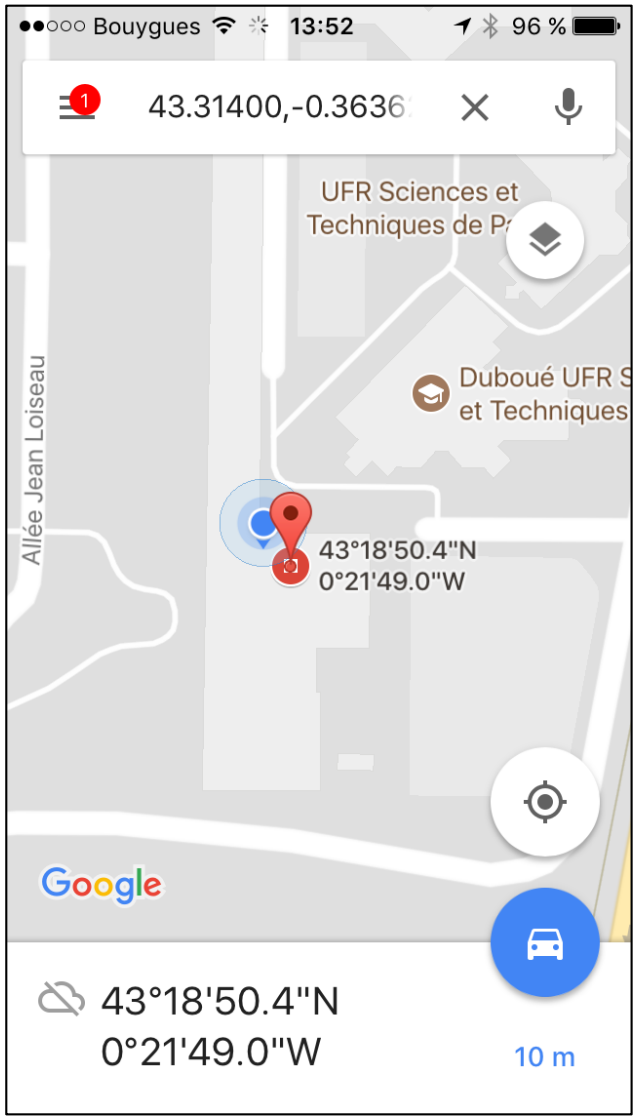
Outdoor tests

```
--- rxlora. dst=1 type=0x12 src=15 seq=188 len=45 SNR=5 RSSIpkt=-90 BW=125 CR=4/5 SF=12
^p1,18,15,188,45,5,-90
^r125,5,12
^t2017-12-19T12:59:34.088
?\!BC/9/LAT/43.31402/LGT/-0.36370/FXT/4180
--- rxlora. dst=1 type=0x12 src=15 seq=189 len=46 SNR=5 RSSIpkt=-92 BW=125 CR=4/5 SF=12
^p1,18,15,189,45,5,-92
^r125,5,12
^t2017-12-19T13:05:23.073
?\!BC/10/LAT/43.31400/LGT/-0.36362/FXT/4172
--- rxlora. dst=1 type=0x12 src=15 seq=190 len=46 SNR=4 RSSIpkt=-91 BW=125 CR=4/5 SF=12
^p1,18,15,190,45,5,-91
^r125,5,12
^t2017-12-19T13:06:12.038
?\!BC/11/LAT/43.31408/LGT/-0.36398/FXT/4866
```

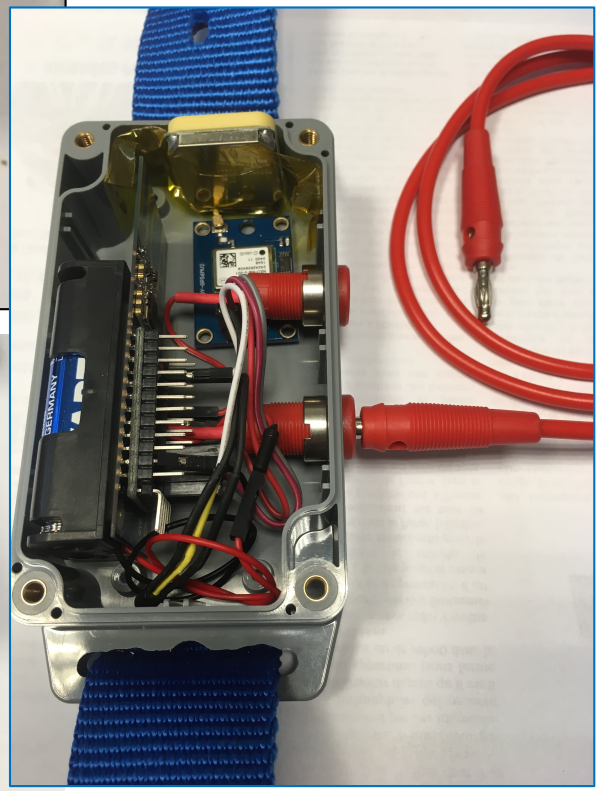
Receiving SMS from gateway

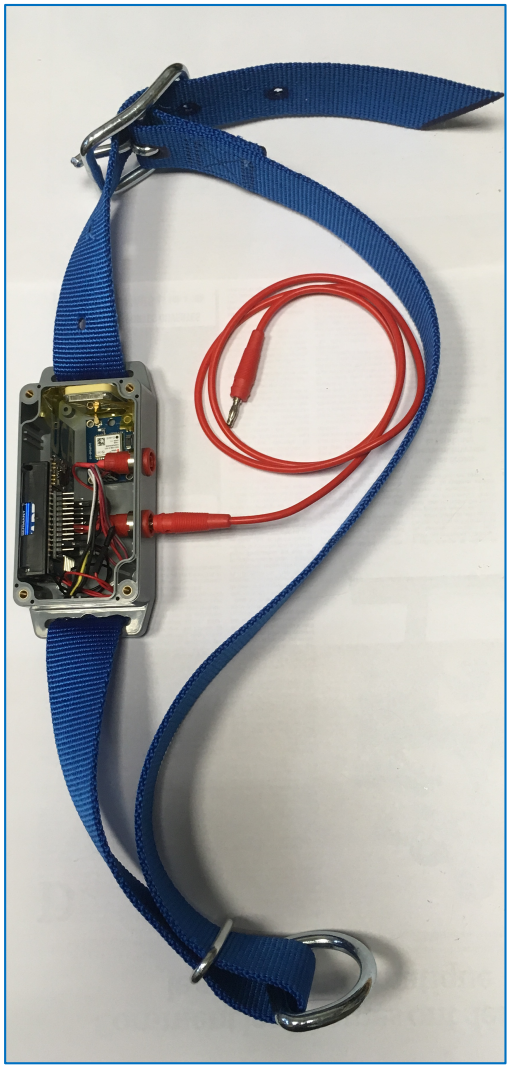
Using the full gateway with post-processing stage and CloudSMS.py enabled (a dongle is needed), you can receive the payload as an SMS on your smartphone





Build a simple collar

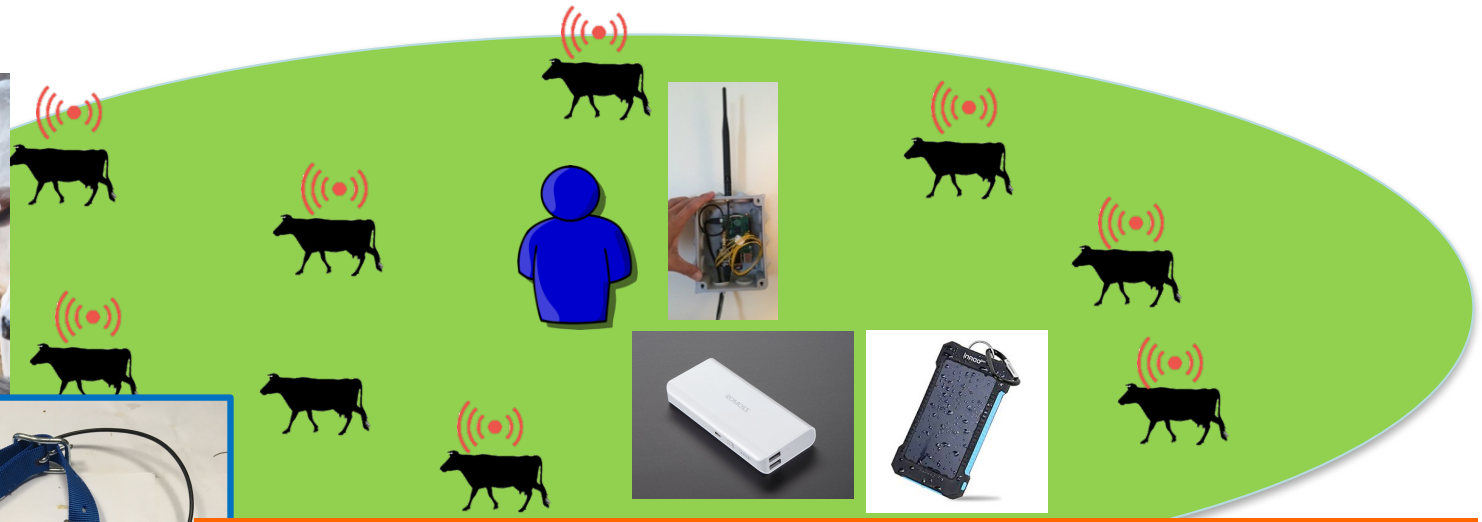




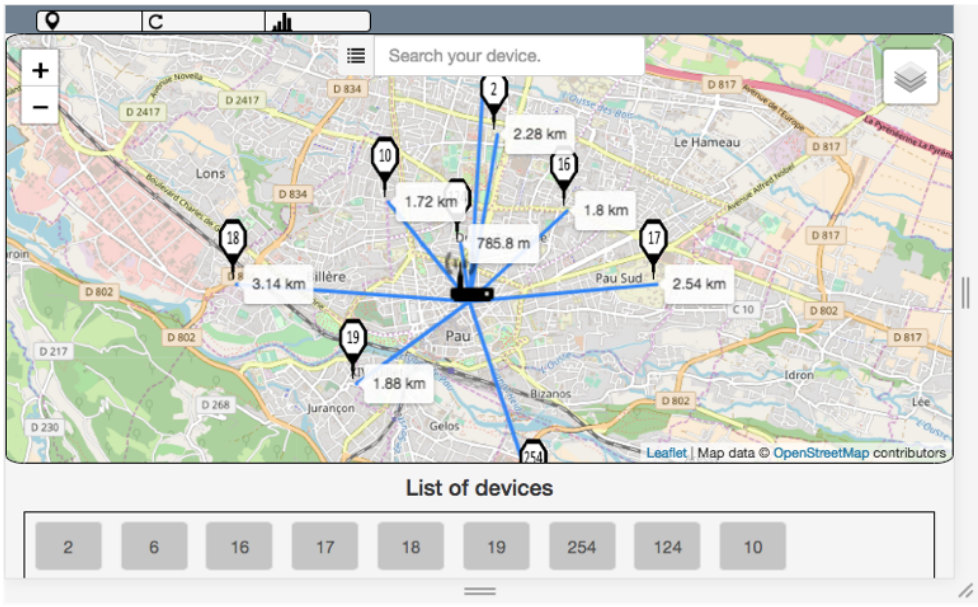
Enlarge the existing holes to pass a belt or a strap

You use a commercial cow collar such as this one from Agro Direct: https://www.agrodirect.fr/collier-numeros/1475-collier-pour-identification-bleu.html, for less than 5€.

MOBILITY SCENARIO WITH AN AUTONOMOUS GATEWAY



A web interface displays the position of the gateway those of the remote GPS devices



Using CloudGpsFile.py

- `CloudGpsFile.py` is a dedicated post-processing module that will search in incoming messages a valid 'LAT' and 'LGT' field such as in "`BC/9/LAT/43.31402/LGT/-0.36370/FXT/4180`"
- You can enable `CloudGpsFile.py` in `clouds.json`. When a message with valid GPS coordinates is received, `CloudGpsFile.py` will write an entry in `gps/gps.txt` file containing relevant packet and GPS information, including the distance (in km) between the gateway and the GPS device

```
src waziup_UPPA_Sensor15 seq 188 bc 9 snr 5 rssi -90 time 2017-11-20T14:18:54 gw
00000027EB5171F7 fxt 4180 lat 43.31402 lgt -0.36370 distance 0.0224
```

- For distance calculation, the gateway position MUST be provided in the `gateway_conf.json` file (see Annex)
- For range test campaign, you can import (or copy/paste) this file in an Excel sheet to plot distance against SNR/RSSI

KEY_GPSFILE.PY

```
#####################################################
#project name
project_name="waziup"

#your organization: CHANGE HERE
#choose one of the following: "DEF", "UPPA", "EGM", "IT21", "CREATENET", "CTIC", "UI", "ISPACE",
"UGB", "WOELAB", "FARMERLINE", "C4A", "PUBD"
organization_name="UPPA"

#sensor name: CHANGE HERE but maybe better to leave it as Sensor
#the final name will contain the sensor address
sensor_name="Sensor"

# CloudGpsFile will built the name as waziup_UPPA_Sensor2
#Note how we can indicate a device source addr that are allowed to use the script
#Use decimal between 2-255 and use 4-byte hex format for LoRaWAN devAddr
#leave empty to allow all devices
#source_list=["3", "255", "01020304"]
source_list=[]

active_interval_minutes=20

gammurc_file="/home/pi/.gammurc"
SMS=False
PIN= "0000"
contacts=["+33XXXXXXXX"]
```

You can change the organization_name.

Set SMS=True if you have a dongle and you want to send SMS on your smartphone to get the calculated distance to the gw

```
src waziup_UPPA_Sensor15 seq 188 bc 9 snr 5 rssi -90 time 2017-11-20T14:18:54+01:00 gw
00000027EB5171F7 fxt 4180 lat 43.31402 lgt -0.36370 distance 0.0224
```

SMS content

MAINTAINING A LIST OF GPS DEVICES (1)

- `CloudGpsFile.py` also maintains a list of GPS devices in `gps/gps.json`

```
{
    "devices": [
        {
            "gw": "00000027EB5171F7",
            "src": "waziup_UPPA_Sensor15",
            "seq": 188,
            "distance": 0.0224,
            "fxt": 4180,
            "bc": 9,
            "lat": 43.31402,
            "snr": 8,
            "time": "2017-11-20T14:18:54",
            "active": "yes",
            "rssi": -45,
            "lgt": -0.3637
        }
    ]
}
```

- New devices (from `src` field) will be added, while existing devices will be updated

Maintaining a list of GPS devices (2)

- `CloudGpsFile.py` also extract from the list of GPS devices those that have sent GPS information in during the last time window
- `key_GpsFile.py` defines
  - `active_interval_minutes=20`
  - For instance, devices that have sent GPS info in the last 20 minutes will be indicated as active
- Those active devices are further maintained in `gps/active_gps.json`
- Further versions can also create kml or gpx file or any combination that would allow more complex visualization features

ANNEXES

GPS NMEA

- GPS coordinates from a GPS module (GNGGA or GPGGA field) are in NMEA format. We convert in decimal degree
  - 0302.78469,N,10601.6986,W
  - 03 => degrees. counted as it is
  - 02.78469 => minutes. divide by 60 before adding to degrees above
  - Hence, the decimal equivalent is: 03 + (02.78469/60) = 3.046412. Multiply by -1 if S
  - For longitude of 10601.6986 => 106+01.6986/60 = 106.02831 degrees. Multiply by -1 if W
- The GPS collar already provides GPS coordinates in decimal degree
  - \!BC/6/LAT/43.31416/LGT/-0.36403/FXT/5833

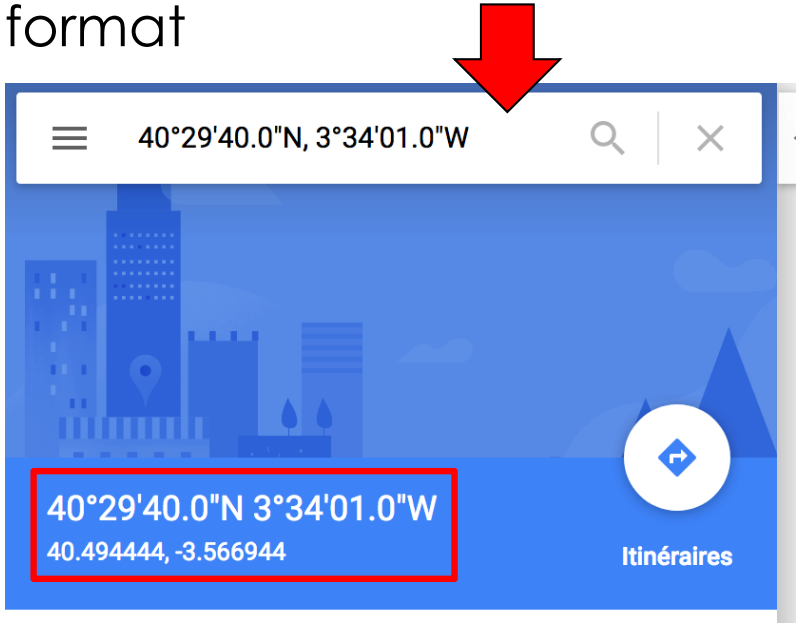
GPS DEGREE,MINUTES,SECONDS

- GPS from smartphone can be indicated in (degree, minutes, second) format
  - 40° 29'40.0"N, 3° 34'01.0"W
- We can convert in decimal degree
  - 40"*100/60=66.666
  - 29'40.0" = 29.66666 => 29.66666/60=0.494444
  - 40° 29'40.0" => 40.494444, positive because N
  - 1"*100/60=1.6666
  - 34'01.0" = 34.16666 => 34.16666/60=0.56944443
  - 3° 34'01.0" => 3.5694443 => -3.5694443 because W

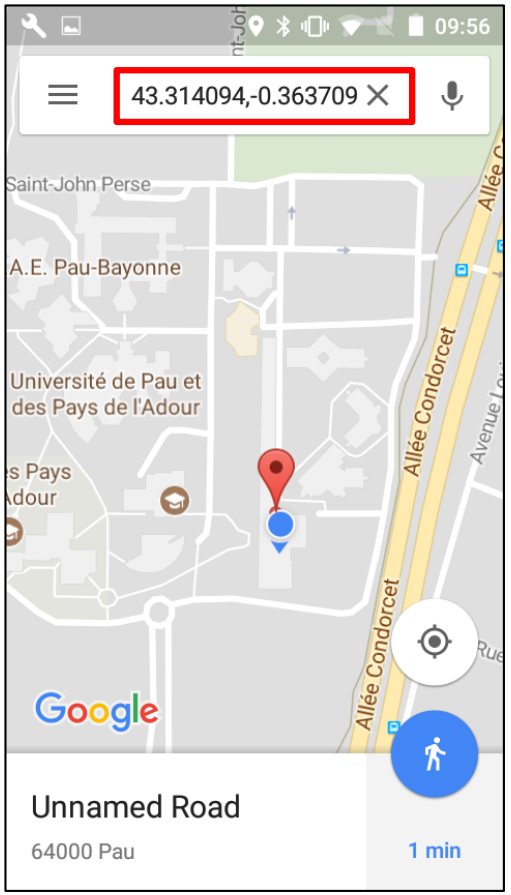
There are many nice GPS tools that give you the GPS coordinates of your location

GPS on GoogleMap

- GoogleMaps accepts GPS coordinates in both degree,minutes,seconds and decimal degree
- If you provide in one format, it will also show the other format

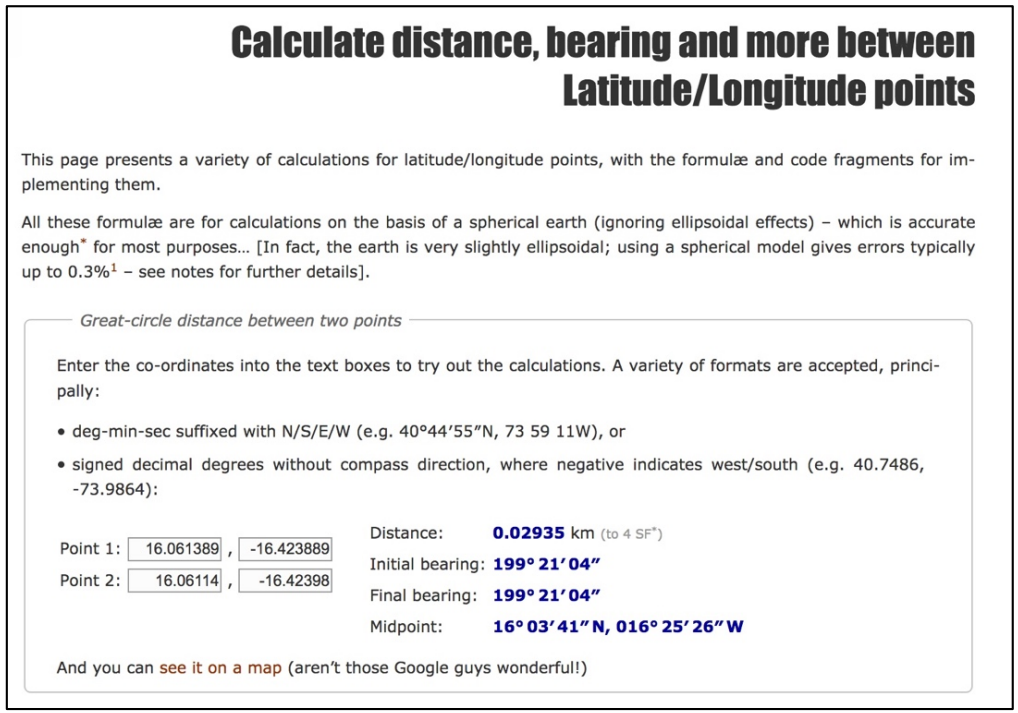


- When you place a pin on a map, GoogleMaps also shows the location in decimal degree



Calculate distance (1)

- Several web sites propose online distance calculation
- https://www.movable-type.co.uk/scripts/latlong.html



Calculate distance, bearing and more between Latitude/Longitude points

This page presents a variety of calculations for latitude/longitude points, with the formulæ and code fragments for implementing them.

All these formulæ are for calculations on the basis of a spherical earth (ignoring ellipsoidal effects) – which is accurate enough* for most purposes… [In fact, the earth is very slightly ellipsoidal; using a spherical model gives errors typically up to 0.3%[1] – see notes for further details].

Great-circle distance between two points

Enter the co-ordinates into the text boxes to try out the calculations. A variety of formats are accepted, principally:

- deg-min-sec suffixed with N/S/E/W (e.g. 40°44′55″N, 73 59 11W), or
- signed decimal degrees without compass direction, where negative indicates west/south (e.g. 40.7486, -73.9864):

Point 1: 16.061389 , -16.423889
Point 2: 16.06114 , -16.42398

Distance: **0.02935** km (to 4 SF*)
Initial bearing: **199° 21′ 04″**
Final bearing: **199° 21′ 04″**
Midpoint: **16° 03′ 41″ N, 016° 25′ 26″ W**

And you can see it on a map (aren't those Google guys wonderful!)

CALCULATE DISTANCE (2)

- `CloudGpsFile.py` calculates the distance (in km) between the gateway and the remote GPS device
- The gateway's coordinates are stored in `gateway_conf.json`
- It uses the haversine function proposed at https://stackoverflow.com/questions/4913349/haversine-formula-in-python-bearing-and-distance-between-two-gps-points

Determining the position of the gateway

- The gateway's coordinates are stored in `gateway_conf.json`

```
"gateway_conf" : {
        "gateway_ID" : "000000XXXXXXXXXX",
        "ref_latitude" : "43.31416",
        "ref_longitude" : "-0.36430",
```

- The simplest way to fill-in these coordinates is
  - to use a smartphone to get the position, see slides 42 using GoogleMap
  - Then to edit `gateway_conf.json` to enter the coordinates

Mobility scenario

- In a mobility scenario such as depicted previously in slide 33 , the position of the gateway can be updated dynamically by plugging a GPS module to the gateway
- The simplest way is to use a USB GPS module which are now quite cheap
- `gateway_conf.json` has a `status_conf` section where `dynamic_gps` can be enabled


```
"status_conf" : {
          "dynamic_gps" : true,
          "gps_port" : "/dev/ttyACM0"
},
```

- The GPS USB/serial port is usually `/dev/ttyACM0` on the RPI3

Testing the USB GPS module (1)

- Plug the USB GPS module to your Raspberry and go to `/home/pi/lora_gateway/sensors_in_raspi`
- Check with `ls -l /dev/tty*` if the port for the new device is `/dev/ttyACM0`

```
...
crw-rw---- 1 root dialout 166,  0 Jan  9 14:43 /dev/ttyACM0
crw-rw---- 1 root dialout 204, 64 Dec 21 12:45 /dev/ttyAMA0
crw------- 1 root root      5,  3 Dec 21 12:45 /dev/ttyprintk
crw-rw---- 1 root dialout   4, 64 Dec 21 12:56 /dev/ttyS0
```

- If it is the case, run

```
python 9600SerialToStdout.py | egrep "GNGGA|GPGGA"
```

- Otherwise run for instance

```
python 9600SerialToStdout.py /dev/ttyUSB0 | egrep
"GNGGA|GPGGA"
```

Testing the USB GPS module (2)

- After some time, you should see something like

```
$GNGGA,142843.00,4318.85982,N,00021.85223,W,1,05,3.87,224.9,M,49.1,M,,*5D
$GNGGA,142844.00,4318.85994,N,00021.85229,W,1,05,3.87,224.9,M,49.1,M,,*57
$GNGGA,142845.00,4318.86012,N,00021.85232,W,1,05,3.87,224.8,M,49.1,M,,*59
$GNGGA,142846.00,4318.86013,N,00021.85233,W,1,05,3.87,224.8,M,49.1,M,,*5A
$GNGGA,142847.00,4318.86025,N,00021.85230,W,1,05,3.87,224.8,M,49.1,M,,*5D
$GNGGA,142848.00,4318.86030,N,00021.85226,W,1,05,3.87,224.7,M,49.1,M,,*5E
$GNGGA,142849.00,4318.86010,N,00021.85225,W,1,05,3.87,224.6,M,49.1,M,,*5F
```

- Which mean that the module has a valid GPS fix. Press CTRL-C to stop
- Then run `python get_gps.py`

```
get_gps.py: removing /home/pi/lora_gateway/gateway_gps.txt file
43.31430,-0.36415
43.31431,-0.36417
43.31432,-0.36417
get_gps.py: saving gps coordinates in /home/pi/lora_gateway/gateway_gps.txt file
```

When enabling dynamic_gps

- Enabling `dynamic_gps` in `gateway_conf.json` activates the following tasks
  - `post_status_processing_gw.py` which is periodically called by `post_processing_gw.py` will try to get the position of the gateway using a connected GPS module. It uses `get_gps.py` in the `sensors_in_raspi` folder
  - `get_gps.py` produces a `gateway_gps.txt` file if a valid GPS fix is obtained. The file simply contains the coordinates in decimal degree: `43.31427,-0.36424`
  - If `post_status_processing_gw.py` finds a `gateway_gps.txt` file, it will update in `gateway_conf.json` the GPS coordinate fields used by `CloudGpsFile.py`