# Sommaire

[**Sommaire 1**](#_t53wabp5skiq)

[**1-Composants présent sur la carte 2**](#_v6x88d85aupq)

[1-1-Raspberry pi pico 2](#_fywzozqhxcow)

[1-2-MPU9250 3](#_6g518lgtahkt)

[1-3-BME680 4](#_nbshxjsugke5)

[1-4-Module gps 4](#_pxn7cvefrfrl)

[**2-Calculs des données 6**](#_mw9pbg9wbchk)

[**3-Choix du module de communication 7**](#_ek6ih16hu7gu)

[HopeRF RFM97W 7](#_saruius9lbpq)

[**4-Codes 8**](#_6b7rj86r81v)

# 

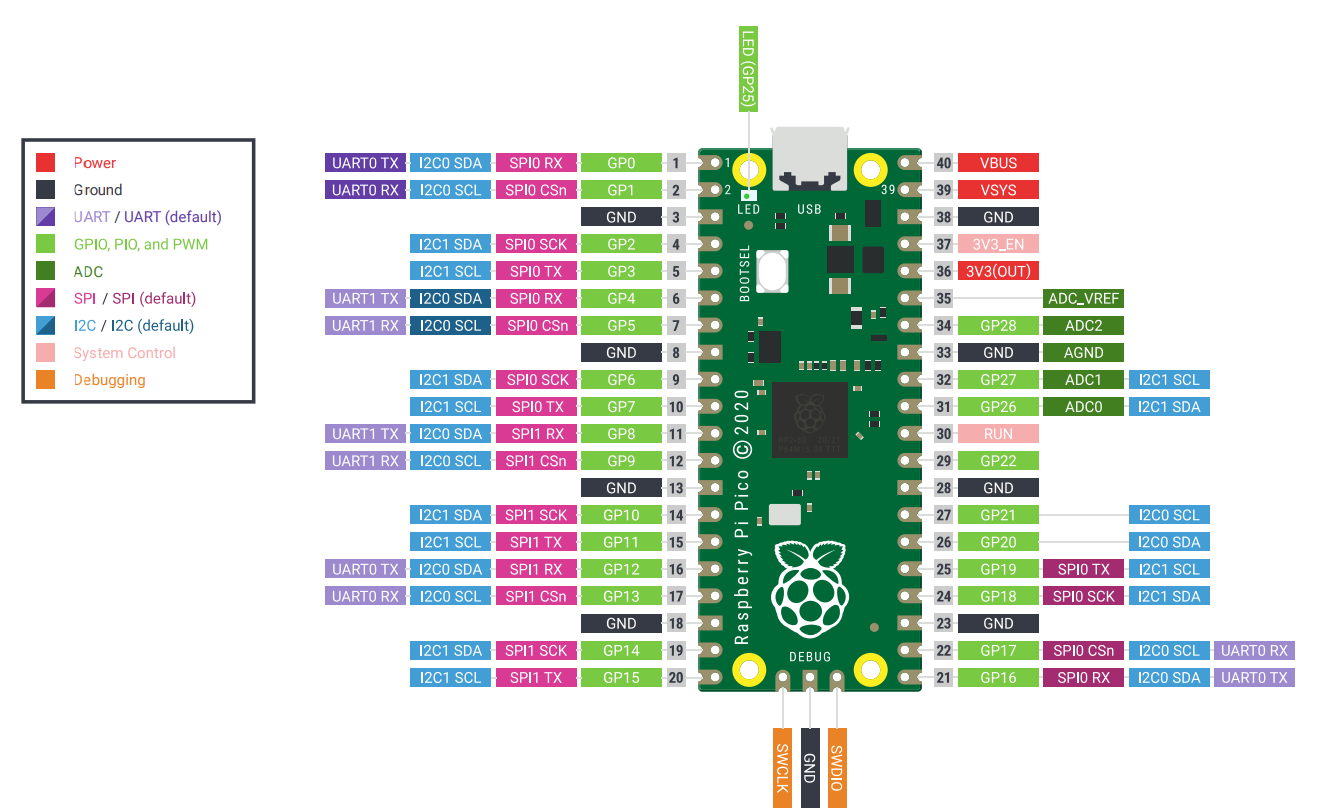
# 1-Composants présent sur la carte

Liste des composants

| Raspberry pi 4 |  |  |  |
| --- | --- | --- | --- |
| MPU9250 | => module GY-91 | AliExpress |  |
| BME680 |  |  |  |
| NEO-M8N | => module | AliExpress |  |

## 1-1-Raspberry pi pico

Pinout diagram :



Datasheet : https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf

La *Raspberry Pi Pico* est équipée d’un quartz à 12 MHz pour l’horloge externe.

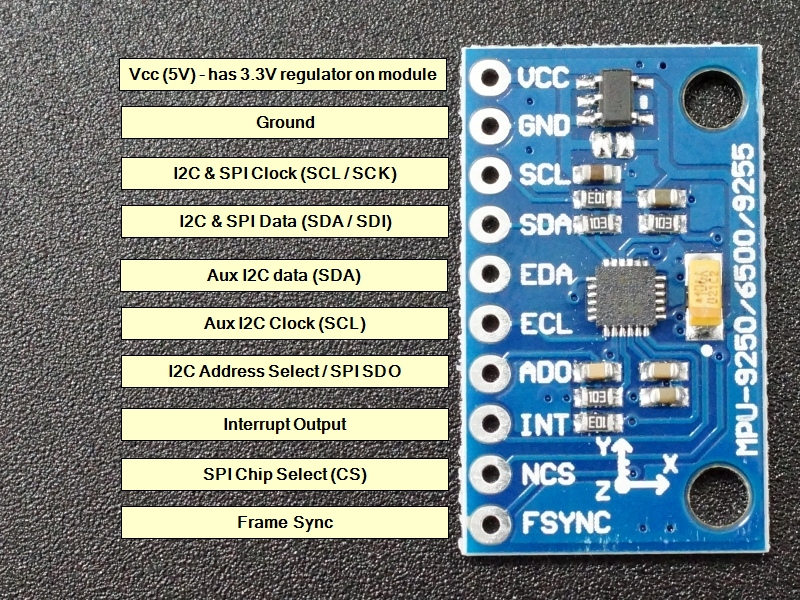


## 1-2-MPU9250 (GY-91)

Datasheet :

https://invensense.tdk.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf

Register map : https://cdn.sparkfun.com/assets/learn\_tutorials/5/5/0/MPU-9250-Register-Map.pdf



### **1. Récupérer les données du GY-91 (MPU9250)**

Le GY-91 te fournit trois types de données :

* **Gyroscope** (de 3 axes)
* **Accéléromètre** (de 3 axes)
* **Magnétomètre** (de 3 axes)

Tu peux utiliser l’**I2C** ou **SPI** pour récupérer ces données, mais I2C est plus courant.

### **2. Fusionner les capteurs**

Pour calculer les quaternions, il te faut **fusionner les données du gyroscope, accéléromètre et magnétomètre**. Plusieurs algorithmes sont disponibles :

* **Algorithme de Madgwick** (léger et efficace pour les microcontrôleurs comme le Raspberry Pi Pico)
* **Algorithme de Mahony** (plus complexe mais également populaire)
* **Filtre de Kalman** (plus coûteux en calcul mais plus précis à long terme)

### **3. Implémentation d'un algorithme simple (Madgwick)**

L'algorithme **Madgwick** est largement utilisé pour des applications embarquées car il est efficace en termes de calcul.

Voici un résumé du processus pour **Madgwick** :

#### **a. Initialiser les variables**

* **Accéléromètre** : normalisé sur 1 (vecteur unitaire).
* **Gyroscope** : intégré pour obtenir l'angle de rotation.
* **Magnétomètre** : pour corriger les dérives magnétiques.

#### **b. Calcul des quaternions**

L'algorithme utilise un filtre à base de **quaternions** pour fusionner les trois sources de données. Tu peux soit utiliser un code déjà prêt (bibliothèque), soit le coder manuellement.

## 1-3-BME680

## 1-4-Module gps

Il faut un module qui fonctionne au moins jusqu’à 5 km. La précision de la localisation doit être inférieure à 4 mètres pour permettre de retrouver les fusées facilement.

**NEO-M8N-0 : 6,40€**

| No. of Channels: | 72 |
| --- | --- |
| Sensitivity: | -167dBm |
| Supply Voltage: | 2.7V to 3.6V |
| Supply Current-Max: | 67mA |
| Cold Capture: | 29s |
| Méthode de montage : | Surface Mount |

https://fr.aliexpress.com/item/4000070604981.html?spm=a2g0o.productlist.main.1.3e54d79ud79uGD&algo\_pvid=3b569d07-20f9-42b7-8c82-d7442f45c66a&algo\_exp\_id=3b569d07-20f9-42b7-8c82-d7442f45c66a-0&pdp\_ext\_f=%7B%22order%22%3A%22180%22%2C%22eval%22%3A%221%22%7D&pdp\_npi=4%40dis%21EUR%211.67%210.99%21%21%211.70%211.01%21%40211b816617377319044215440e5592%2110000000181416176%21sea%21FR%210%21ABX&curPageLogUid=IwSDMNpJA3uW&utparam-url=scene%3Asearch%7Cquery\_from%3A

**Specification:**

Antenna Size: approx. 25\*25mm/0.98\*0.98"

Module Size: approx. 25\*36\*6mm/0.98\*1.41\*0.23"

Net weight: approx. 16g

Gross weight: approx. 17g

Mounting: approx. 3mm

The default baud rate: 9600

**Module specs:**

Module with ceramic destined antenna, super signal

EEPROM power down to save the configuration parameters data

LED signal indicator

With data backup battery

The default baud rate: 9600

Mounting Hole 3mm

Module size 23mm \* 30mm

Antenna size 12 \* 12mm

Cable:20mm

**Features:**

* GPS modules NEO-7M, 3V-5V power supply
* Module with ceramic destined antenna, signal super
* With data backup battery
* High quality Immersion Gold PCB
* Module Size: 25\*35mm/0.98\*1.37"
* Supply Voltage: 3V-5V
* Using imported NEO-7M-0-000 GPS module
* EEPROM power down to save the configuration parameter data
* There are LED lights
* Antenna Size: 25\*25mm/0.98\*0.98"
* Installation Diameter: 3mm
* The default baud rate: 9600

# 2-Calculs des données

**Trame du GY-91 => 24 octets (Fmax)**

Gyroscope (8khz) : 3 \* 2 octets = 6 octets

Accelerometer (1Khz) : 3 \* 2 octets = 6 octets

Magnetometer (100 Hz) : 3 \* 2 octets = 6 octets

Pressure (157 Hz) : 2 octets

Temperature (157 Hz) : 2 octets

Or avec le gy91 on peut envoyer des quaternions (16 octets en activant le DMP) “askip”. Ce qui reviendrait à 20 octets au total ce qui est mieux que 24 octets

**Trame du BME680 => 12 octets**

Température (50 Hz) : 20 bits

Pression (50 Hz) : 20 bits

Humidité (25 Hz) : 20 bits

Gaz (1 Hz) : 16 - 24 bits : 2 - 3 octets

Calcul du bps max : Frequence \* Nombre\_Octets \* 8

8000\*6\*8 + 1000\*6\*8 + 100\*6\*8 + 157\*16\*2 = 441824 bit/s = 55228 octets/s = 441 Kbps

Ainsi nous devons prévoir un module de communication qui pourra atteindre cette vitesse de transmission. De même nous devons nous assurer que la carte raspberry puisse stocker ces données. Malheureusement, les contraintes sont trop élevées, il faut alors optimiser les données que l’on veut envoyer.

Le calcul suivant prend en compte tous les capteurs qui envoient des données avec le LoRa. Il y a un peu d’optimisation au niveau des fréquences mais pour le GY-91 on envoie tout (pas les quaternions donc 24 octets). Et le BME680 est en mode gaz (trame => 12 octets).

6\*8( 50 + 25 + 25) + 1\*2\*16 + 12\*8\*1 = 4928 bps = 4,928 Kbps < 5,47 Kbps

On peut donc remonter légèrement les fréquences pour le gyro

6\*8( 60 + 25 + 25) + 1\*2\*16 + 12\*8\*1 = 5408 bps = 5,408 Kbps < 5,47 Kbps

Pour utiliser du LoRa il faut rester en dessous de 5,47 Kbps (pour rester en SF7 , 5Km).

Calcul du stockage :

Si la fusée est en fonctionnement pendant 10 min.

10\*60\*55228 = 33 136 800 octets = 33,13 Mo de stockage nécessaire.

# 3-Choix du module de communication

Cependant il existe des plage de fréquence interdite par Planète Science :

- Inférieure à 10mW pour la bande de fréquence 433.05MHz à 434.79MHz ;

- Inférieure à 25mW pour la bande de fréquence 868MHz à 869.2MHz ;

- Inférieure à 500mW pour la bande de fréquence 869.4MHz à 869.65MHz.

### HopeRF RFM95W

<https://fr.aliexpress.com/item/1005003249965079.html?spm=a2g0o.productlist.main.1.4db4E0RYE0RYAa&algo_pvid=02e1cf12-fc3c-4b86-9b29-a602f33fadc8&algo_exp_id=02e1cf12-fc3c-4b86-9b29-a602f33fadc8-0&pdp_ext_f=%7B%22order%22%3A%22359%22%2C%22eval%22%3A%221%22%7D&pdp_npi=4%40dis%21EUR%212.85%210.99%21%21%213.00%211.04%21%40211b80c217429881577562389ea43a%2112000024855316430%21sea%21FR%210%21ABX&curPageLogUid=7k6MFRAvbzqu&utparam-url=scene%3Asearch%7Cquery_from%3A>

868/915 MHz 6 - 9 7.8 - 500 kHz 0.11 - 37.5 kbps -111 to -139 dBm

* Puissance Tx : Jusqu'à 20 dBm (configurable à 14 dBm pour 25 mW).
* Fréquence : 868 MHz.
* Débit RF : Jusqu'à 37.5 kbps en LoRa.
* Portée : Jusqu'à 10 km en champ libre (à 10 kbps avec bonne antenne).
* Interface : SPI (à connecter à un microcontrôleur comme STM32 ou Arduino).
* Taille compacte : Idéal pour des projets embarqués.
* Prix : Environ 4-8 €, largement disponible sur des sites comme Mouser, Digikey ou AliExpress.

### **Débit en fonction de la distance (5 km) et puissance < 500 mW**

* En LoRa, le débit dépend du **Spreading Factor (SF)** et de la **bande passante (BW)**.
* À **869,4 MHz**, l’**ETSI** impose un duty cycle de **10%** (région Europe).
* En général, à **5 km**, on utilise **SF10 ou SF12** pour assurer une bonne réception.

#### **Exemples de débits possibles (LoRa, 125 kHz BW)**

| **Spreading Factor** | **Débit max (~bps)** |
| --- | --- |
| **SF7** | ~5.4 kbps |
| **SF8** | ~3 kbps |
| **SF9** | ~1.7 kbps |
| **SF10** | ~980 bps |
| **SF11** | ~440 bps |
| **SF12** | ~250 bps |

À **5 km**, pour rester fiable, il faut viser **SF10-SF12**, donc un débit de **250-980 bps** environ.

Si tu as besoin d’un débit plus élevé, tu peux :

* Augmenter la bande passante (ex: **250 kHz** → x2 en débit).
* Réduire la portée (permet d'utiliser SF plus bas).
* Optimiser l’antenne et le positionnement.

# 4-Codes

CODE HTML pour réception

<!DOCTYPE html>

<html>

<head>

<title>État du Bouton Arduino</title>

<meta charset='UTF-8'> <!-- Définit l'encodage des caractères en UTF-8 -->

</head>

<body>

<h1>État de la connexion LORA</h1>

<p>État actuel : <span id='status'>Non connecté</span></p> <!-- Affiche l'état actuel du bouton -->

<button id='connectBtn'>Connecter Arduino</button> <!-- Bouton pour établir la connexion avec l'Arduino -->

<script>

// Récupère les éléments HTML dans des constantes pour manipulation

const statusElement = document.getElementById('status'); // Élément où afficher l'état du bouton

const connectBtn = document.getElementById('connectBtn'); // Bouton de connexion

// Fonction asynchrone pour établir la connexion avec l'Arduino

async function connectLORA() {

try {

// Demande à l'utilisateur de sélectionner un port série

const port = await navigator.serial.requestPort();

// Ouvre le port série sélectionné avec une vitesse en bauds correspondant à celle configurée sur l'Arduino

await port.open({ baudRate: 115200 });

// Crée un lecteur pour recevoir les données du port série

const reader = port.readable.getReader();

// Désactive le bouton pour empêcher de multiples connexions

connectBtn.disabled = true;

// Change l'état affiché pour indiquer que la connexion est établie

statusElement.textContent = 'Connecté';

// Décodeur pour convertir les données brutes reçues en texte

const decoder = new TextDecoder();

// Boucle infinie pour lire les données envoyées par l'Arduino

while (true) {

// Lit les données depuis le port série

const { value, done } = await reader.read();

// Si la lecture est terminée, sort de la boucle

if (done) break;

// Décode les données reçues en texte et enlève les espaces vides

const decodedValue = decoder.decode(value).trim();

// Affiche la valeur reçue dans la console pour le débogage

console.log("Reçu :", decodedValue);

// Met à jour l'état affiché selon la valeur reçue

if (decodedValue === '0') { // Si l'Arduino envoie '0'

statusElement.textContent = 'Bouton PRESSÉ';

} else if (decodedValue === '1') { // Si l'Arduino envoie '1'

statusElement.textContent = 'Bouton NON pressé';

} else {

// Si la valeur reçue est inattendue, affiche un avertissement

console.warn("Valeur inattendue :", decodedValue);

}

}

} catch (error) {

// En cas d'erreur, affiche le message dans la console et met à jour l'état

console.error('Erreur de connexion :', error);

statusElement.textContent = 'Erreur de connexion';

connectBtn.disabled = false; // Réactive le bouton pour réessayer

}

}

// Ajoute un écouteur d'événement au bouton pour exécuter la fonction `connectArduino` lorsqu'on clique dessus

connectBtn.addEventListener('click', connectArduino);

</script>

</body>

</html>

**CODE arduino pour la carte communication :**

#include "MPU9250.h"

#include <SoftwareSerial.h>

#include <Wire.h>

// Define SoftwareSerial pins for communication with E32

#define E32\_RX 10 // Arduino receives data from E32

#define E32\_TX 11 // Arduino sends data to E32

#define M0 8 // Mode selection pin

#define M1 9 // Mode selection pin

// Définir les broches pour I2C (par défaut, SDA = GP0 et SCL = GP1 pour I2C0)

#define SDA\_PIN 1

#define SCL\_PIN 2

SoftwareSerial e32Serial(E32\_RX, E32\_TX); // RX, TX

MPU9250 mpu;

void setup() {

Serial.begin(57600);

e32Serial.begin(9600);

// Set pin modes for M0 and M1

pinMode(M0, OUTPUT);

pinMode(M1, OUTPUT);

// Set module to normal mode (M0 = LOW, M1 = LOW)

digitalWrite(M0, LOW);

digitalWrite(M1, LOW);

//--------------------------------------------------------------------------MPU920-------------------------------------------------------------------

Wire.begin();

delay(2000);

if (!mpu.setup(0x68)) { // if AD0 grounded

while (1) {

Serial.println("MPU connection failed. Please check your connection with `connection\_check` example.");

delay(5000);

}

}

// calibrate anytime you want to

Serial.println("Accel Gyro calibration will start in 5sec.");

Serial.println("Please leave the device still on the flat plane.");

mpu.verbose(true);

delay(5000);

mpu.calibrateAccelGyro();

Serial.println("Mag calibration will start in 5sec.");

Serial.println("Please Wave device in a figure eight until done.");

delay(5000);

mpu.calibrateMag();

print\_calibration();

mpu.verbose(false);

}

void loop() {

if (mpu.update()) {

static uint32\_t prev\_ms = millis();

if (millis() > prev\_ms + 25) {

if (Serial.available()) {

String result = "Yaw: " + String(mpu.getYaw(), 2) + ", Pitch: " + String(mpu.getPitch(), 2) + ", Roll: " + String(mpu.getRoll(), 2);

e32Serial.print(result); // Send to E32

Serial.println("Sent to E32: " + result);

}

print\_roll\_pitch\_yaw();

prev\_ms = millis();

}

}

}

void print\_roll\_pitch\_yaw() {

Serial.print("Yaw, Pitch, Roll: ");

Serial.print(mpu.getYaw(), 2);

Serial.print(", ");

Serial.print(mpu.getPitch(), 2);

Serial.print(", ");

Serial.println(mpu.getRoll(), 2);

}

void print\_calibration() {

Serial.println("< calibration parameters >");

Serial.println("accel bias [g]: ");

Serial.print(mpu.getAccBiasX() \* 1000.f / (float)MPU9250::CALIB\_ACCEL\_SENSITIVITY);

Serial.print(", ");

Serial.print(mpu.getAccBiasY() \* 1000.f / (float)MPU9250::CALIB\_ACCEL\_SENSITIVITY);

Serial.print(", ");

Serial.print(mpu.getAccBiasZ() \* 1000.f / (float)MPU9250::CALIB\_ACCEL\_SENSITIVITY);

Serial.println();

Serial.println("gyro bias [deg/s]: ");

Serial.print(mpu.getGyroBiasX() / (float)MPU9250::CALIB\_GYRO\_SENSITIVITY);

Serial.print(", ");

Serial.print(mpu.getGyroBiasY() / (float)MPU9250::CALIB\_GYRO\_SENSITIVITY);

Serial.print(", ");

Serial.print(mpu.getGyroBiasZ() / (float)MPU9250::CALIB\_GYRO\_SENSITIVITY);

Serial.println();

Serial.println("mag bias [mG]: ");

Serial.print(mpu.getMagBiasX());

Serial.print(", ");

Serial.print(mpu.getMagBiasY());

Serial.print(", ");

Serial.print(mpu.getMagBiasZ());

Serial.println();

Serial.println("mag scale []: ");

Serial.print(mpu.getMagScaleX());

Serial.print(", ");

Serial.print(mpu.getMagScaleY());

Serial.print(", ");

Serial.print(mpu.getMagScaleZ());

Serial.println();

}

# 5 - Prototypage

## 5.1 - Test du GPS

Pour le test du GPS (Neo6m) : <https://youtu.be/CLsXnSOIYMg>

Après avoir écrit un code simple de lecture en UART on obtient des données sous la forme :

$GPRMC,,V,,,,,,,,,,N\*53

$GPVTG,,,,,,,,,N\*30

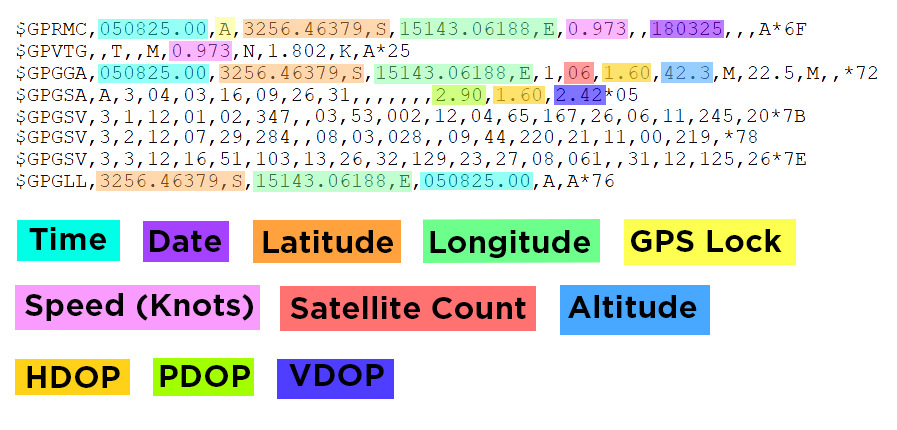
$GPGGA,,,,,,0,00,99.99,,,,,,\*48

$GPGSA,A,1,,,,,,,,,,,,,99.99,99.99,99.99\*30

$GPGSV,1,1,00\*79

$GPGLL,,,,,,V,N\*64

On remarque que le “V” dans la première ligne indique que le GPS n’a pas encore réussi à se ‘caler’ sur un satellite. Après que le GPS est “lock” sur un satellite on obtient alors des données de la forme :



Dans notre cas, tout ce qui nous intéresse, c’est la latitude et la longitude et en ‘bonus’ l’altitude.