Guide d’utilisation



Une image contenant câble, Appareils électroniques, fils électriques, Ingénierie électronique

Le contenu généré par l’IA peut être incorrect.

Table des matières

[Branchement des composants 3](#_Toc193095793)

[Premiers pas sur Arduino 4](#_Toc193095794)

[Installation du Framework et de la Library 4](#_Toc193095795)

[Création de notre projet 5](#_Toc193095796)

[Explication détaillée du code Arduino pour l’affichage de la distance 5](#_Toc193095797)

[1. Inclusion des bibliothèques : 5](#_Toc193095798)

[2. Déclaration des variables et initialisation : 5](#_Toc193095799)

[3. Initialisation de l’écran OLED: 5](#_Toc193095800)

[4. Contrôle de l’alimentation de l’écran OLED : 6](#_Toc193095801)

[5. Fonction pour mesurer la distance avec le capteur LiDAR : 6](#_Toc193095802)

[6. Initialisation du programme (*setup()*). 7](#_Toc193095803)

[7. Boucle principale (*loop()*). 8](#_Toc193095804)

[Envoie des données Lora sur la passerelle LoraWan 10](#_Toc193095805)

[1. Initialisation de LoRaWAN : 10](#_Toc193095806)

[2. Configuration des paramètres LoRaWAN : 10](#_Toc193095807)

[3. Envoi des données vers la passerelle : 11](#_Toc193095808)

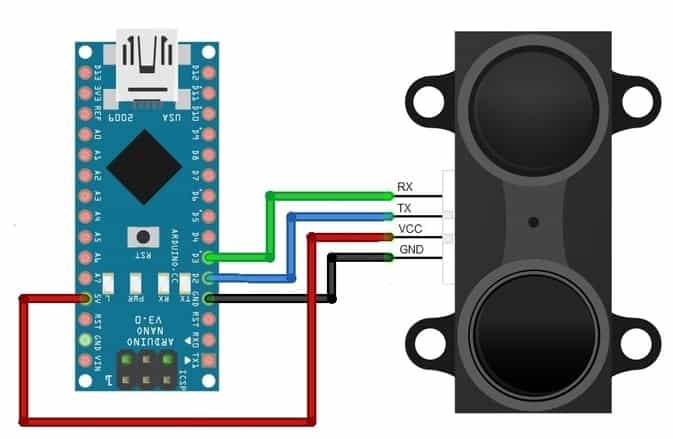
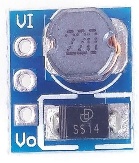
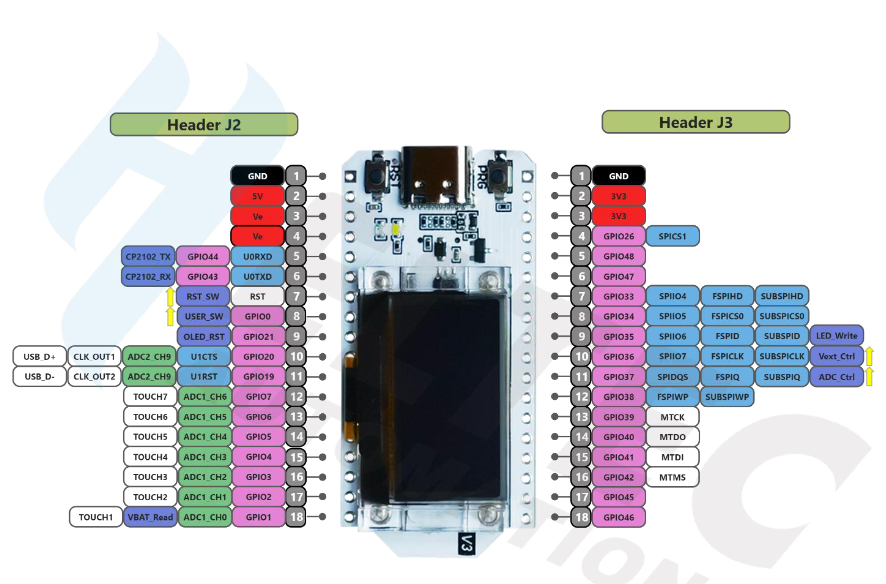
[4. Envoi via LoRaWAN : 11](#_Toc193095809)

[5. Mise en veille et réveil : 12](#_Toc193095810)

[6. Résumé du fonctionnement LoRaWAN : 13](#_Toc193095811)

[Conclusion 14](#_Toc193095812)

# Branchement des composants



**Comment la tension passe de 3V3 en 5V :**

**Etape 1 :**

Branchement 3V3 (fil rouge), connecter à la broche VI.

**Etape 2 :**

Branchement GND (fil noir), connecter au GND de l’élévateur de tension.

**Etape 3 :**

Branchement V0 (fil rouge), connecter à la tension (courant continu) du Lidar.

**Etape 4 :**

Branchement GND (fil noir), au Lidar Garmin.

**Communication I2C**

* Broche SDA (fil bleu), connecter à la broche GPIO41.
* Broche SCL (fil vert), connecter à la broche GPIO42.

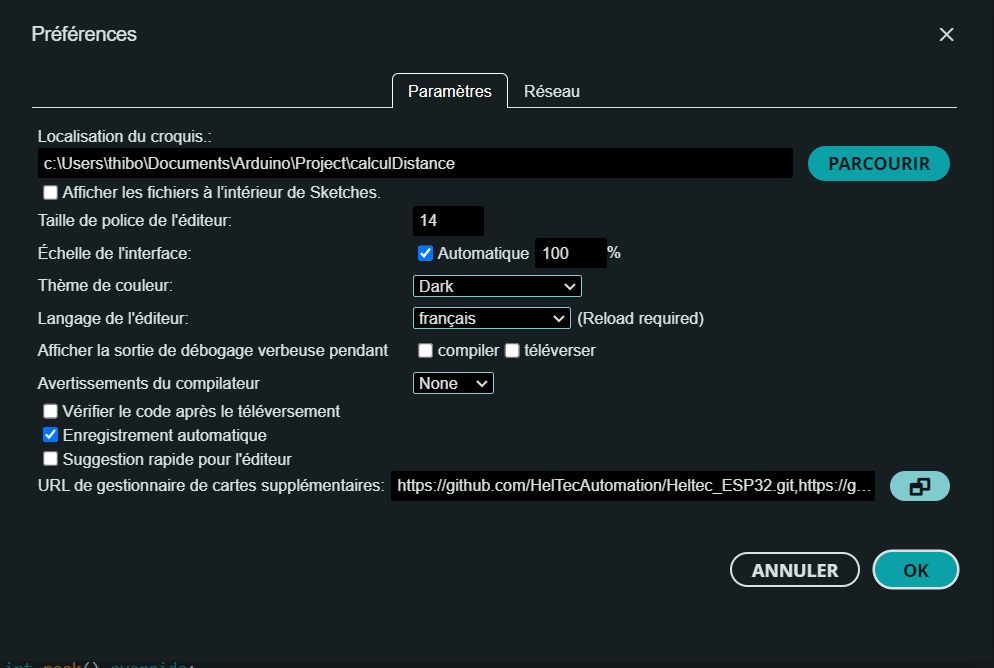
# Premiers pas sur Arduino

## Installation du Framework et de la Library

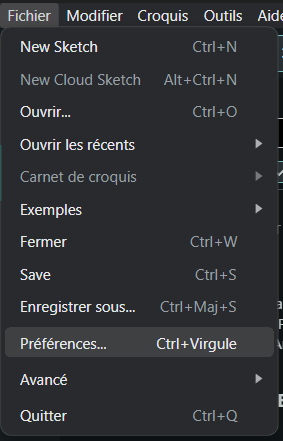
Dans un premier temps nous allons installer le Framework et la bibliothèque de la carte ESP32 Heltec V3 sur Arduino.

À savoir :

* Un Framework permet de contrôler les appareils connecter.
* Les librairies permettant de simplifier l'utilisation d'un capteur ou d'une fonctionnalité.



**Etape 1 :**



*Fichier* =>*Préférences.*

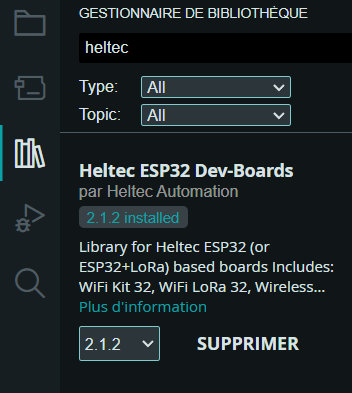
**Etape 2 :**

Copier-coller l’URL du package : <https://resource.heltec.cn/download/package_heltec_esp32_index.json>

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.Nous pouvons ensuite installer dans le gestionnaire de carte, le Framework :

Pour l’installation de la bibliothèque, nous nous rendons dans le gestionnaire de bibliothèques :



# Création de notre projet

Une fois le branchement et la vérification des composants, nous allons pouvoir commencer le but de notre projet.

## Explication détaillée du code Arduino pour l’affichage de la distance

### 1. Inclusion des bibliothèques :

#include <Wire.h>

#include "HT\_SSD1306Wire.h"

* *Wire.h* : Bibliothèque pour la communication I2C (utilisée pour le capteur LiDAR et l'écran OLED).
* *HT\_SSD1306Wire.h* : Bibliothèque pour piloter l’écran OLED SSD1306.

### 2. Déclaration des variables et initialisation :

extern TwoWire Wire1;

int SDA\_PIN = 41;

int SCL\_PIN = 42;

#define LIDAR\_ADDR 0x62

* *Wire1* : Déclare un objet pour la communication I2C sur un bus secondaire.
* *SDA\_PIN* et *SCL\_PIN* : Définit les broches utilisées pour la communication I2C avec le capteur LiDAR.
* LIDAR\_ADDR : Adresse I2C du capteur LiDAR (*0x62*).

### 3. Initialisation de l’écran OLED:

SSD1306Wire factory\_display(0x3c, 500000, SDA\_OLED, SCL\_OLED, GEOMETRY\_128\_64, RST\_OLED);

Initialise un écran OLED *SSD1306* avec :

* Adresse I2C *0x3C*
* Vitesse de communication *500000*
* Broches *SDA\_OLED* et *SCL\_OLED* pour l’I2C
* Résolution *128x64* pixels
* Broche *RST\_OLED* pour la réinitialisation

### 4. Contrôle de l’alimentation de l’écran OLED :

*void VextOn(void) {*

*pinMode(Vext, OUTPUT);*

*digitalWrite(Vext, LOW);*

*}*

*void VextOff(void) {*

*pinMode(Vext, OUTPUT);*

*digitalWrite(Vext, HIGH);*

*}*

* *VextOn()* : Active l'alimentation de l'écran OLED.
* *VextOff()* : Désactive l’alimentation de l’écran OLED.

### 5. Fonction pour mesurer la distance avec le capteur LiDAR :

*int getLidarDistance() {*

*Wire1.beginTransmission(LIDAR\_ADDR);*

*Wire1.write(0x00);* ***// Commande pour déclencher une mesure***

*Wire1.write(0x04);* ***// Mode distance avec correction***

*int error = Wire1.endTransmission();*

*if (error != 0) {*

*Serial.println("Erreur : Échec de l'envoi de la commande au LiDAR !");*

*return -1;*

*}*

*delay(500);* ***// Temps pour obtenir une mesure***

*Wire1.beginTransmission(LIDAR\_ADDR);*

*Wire1.write(0x8F);* ***// Registre contenant la distance***

*error = Wire1.endTransmission(false);*

*if (error != 0) {*

*Serial.println("Erreur : Impossible d’accéder au registre de distance !");*

*return -1;*

*}*

*Wire1.requestFrom(LIDAR\_ADDR, 2);*

*if (Wire1.available() >= 2) {*

*int highByte = Wire1.read();*

*int lowByte = Wire1.read();*

*return (highByte << 8) | lowByte;*

*} else {*

*Serial.println("Erreur : Pas assez de données reçues du LiDAR !");*

*return -1;*

*}*

*}*

* Envoie une commande au capteur LiDAR pour mesurer la distance.
* Attente de 500 ms pour laisser le capteur effectuer la mesure.
* Lit la valeur mesurée dans le registre *0x8F* (registre de distance).
* Retourne la distance en combinant les deux octets reçus (*highByte* et *lowByte*).
* En cas d’erreur, affiche un message sur le moniteur série et retourne **-1**.

### 6. Initialisation du programme (*setup()*).

*void setup() {*

*VextOn();* ***// Active l'écran OLED***

*delay(100);*

*factory\_display.init();*

*factory\_display.clear();*

*factory\_display.setFont(ArialMT\_Plain\_24);*

*factory\_display.setTextAlignment(TEXT\_ALIGN\_LEFT);*

*factory\_display.display();*

*pinMode(LED ,OUTPUT);*

*digitalWrite(LED, LOW);*

*Wire.begin(SDA\_OLED, SCL\_OLED);*

*Wire1.begin(SDA\_PIN, SCL\_PIN, 400000);*

*Wire1.beginTransmission(LIDAR\_ADDR);*

*Wire1.write(0x62);*

*Wire1.write(0);*

*Serial.begin(115200);*

*}*

* Active l’écran OLED avec *VextOn()*.
* Initialise l’écran OLED (*factory\_display.init()*).
* Définit l’affichage avec une police et un alignement.
* Configure la LED en sortie (*pinMode(LED, OUTPUT)*).
* Initialise la communication I2C pour l’écran et le capteur LiDAR.
* Démarre la communication série à **115200 bauds**.

### 7. Boucle principale (*loop()*).

void loop() {

int distance = getLidarDistance();

**// Affichage sur le moniteur série**

if (distance >= 0) {

Serial.print("Distance mesurée: ");

Serial.print(distance);

Serial.println(" cm");

} else {

Serial.println("Erreur de lecture LiDAR");

}

factory\_display.clear();

factory\_display.display();

factory\_display.setTextAlignment(TEXT\_ALIGN\_CENTER);

if (distance >= 0) {

factory\_display.drawString(64, 25, String(distance) + " cm");

} else {

factory\_display.setTextAlignment(TEXT\_ALIGN\_CENTER);

factory\_display.setFont(ArialMT\_Plain\_10);

factory\_display.drawString(64, 25, "Erreur de lecture LiDAR");

}

factory\_display.display();

delay(1000);

}

* Appelle *getLidarDistance()* pour mesurer la distance.
* Affiche la distance mesurée sur le **moniteur série**.
* Efface l’écran OLED et met à jour l'affichage :
  + Si la mesure est valide, affiche la distance en cm.
  + Sinon, affiche un message d'erreur.
* Attente de 1 seconde (*delay(1000)*) avant de recommencer.

Voici le résultat final :

Une image contenant câble, Appareils électroniques, fils électriques, Ingénierie électronique

Le contenu généré par l’IA peut être incorrect.

## Envoie des données Lora sur la passerelle LoraWan

### 1. Initialisation de LoRaWAN :

**Déclarations des paramètres OTAA et ABP**

Le code supporte deux modes d'activation pour LoRaWAN :

* **OTAA (Over The Air Activation)** → Recommandé pour la sécurité et l'évolutivité.
* **ABP (Activation By Personalization)** → Moins sécurisé mais plus simple.

Les clés nécessaires sont définies :

uint8\_t devEui[] = { 0x70, 0xB3, 0xD5, 0x7E, 0xD0, 0x06, 0x53, 0xC8 };

uint8\_t appEui[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };

uint8\_t appKey[] = { 0x74, 0xD6, 0x6E, 0x63, 0x45, 0x82, 0x48, 0x27, 0xFE, 0xC5, 0xB7, 0x70, 0xBA, 0x2B, 0x50, 0x45 };

* devEui: Identifiant unique du module LoRaWAN.
* appEui: Identifiant de l'application LoRaWAN.
* appKey: Clé de chiffrement pour la session.

### 2. Configuration des paramètres LoRaWAN :

LoRaMacRegion\_t loraWanRegion = ACTIVE\_REGION; **// Région LoRaWAN (EU868)**

DeviceClass\_t loraWanClass = CLASS\_A; **// Classe du dispositif (A = moins énergivore)**

bool overTheAirActivation = true; **// OTAA activé**

bool loraWanAdr = true; **// Activation de l'Adaptative Data Rate (ADR)**

bool isTxConfirmed = true; **// Envoi des messages confirmés**

uint8\_t appPort = 2; **// Port applicatif pour l'envoi des données**

uint8\_t confirmedNbTrials = 4; **// Nombre de tentatives en cas d'échec**

* loraWanRegion : Définition de la fréquence utilisée (Europe = EU868).
* loraWanClass : Définition de la classe du périphérique (A, B, ou C).
* overTheAirActivation : Active ou désactive OTAA.
* loraWanAdr : Permet au réseau d'ajuster automatiquement le débit de données.
* isTxConfirmed : Indique si l'appareil doit attendre une confirmation de la passerelle.
* appPort : Spécifie le port utilisé pour envoyer les données.

Cycle d’envoie des données :

uint32\_t appTxDutyCycle = 2000; **// Temps entre deux envois (2 secondes)**

### 3. Envoi des données vers la passerelle :

**Construction du message**

Le message envoyé est la distance mesurée par le LiDAR. Il est préparé dans prepareTxFrame() :

static void prepareTxFrame( uint8\_t port )

{

appDataSize = 2; **// Taille du message (2 octets)**

appData[0] = (dst >> 8) & 0xFF; **// Octet de poids fort**

appData[1] = dst & 0xFF; **// Octet de poids faible**

Serial.print("Payload LoRaWAN : ");

Serial.print(appData[0], HEX);

Serial.print(" ");

Serial.println(appData[1], HEX);

}

* Les 2 octets représentent la distance en centimètres sous forme de nombre 16 bits (big-endian).
* dst contient la distance récupérée du LiDAR.

### 4. Envoi via LoRaWAN :

Dans la boucle principale (loop()), l'état du périphérique LoRaWAN est géré par une machine à états :

switch(deviceState )

Les états les plus importants :

1. **DEVICE\_STATE\_INIT** → Initialise LoRaWAN et sélectionne le mode de connexion.
2. **DEVICE\_STATE\_JOIN** → L'appareil tente de rejoindre le réseau.
3. **DEVICE\_STATE\_SEND** → Envoie la mesure de distance au serveur via LoRaWAN.
4. **DEVICE\_STATE\_CYCLE** → Planifie l'envoi du prochain message après un certain délai.
5. **DEVICE\_STATE\_SLEEP** → Met l'appareil en veille pour économiser de l'énergie.

Le passage entre les états est géré par :

case DEVICE\_STATE\_SEND:

{

distanceCheck(); **// Mesure la distance avec le LiDAR**

prepareTxFrame(appPort); **// Prépare le message**

LoRaWAN.send(); **// Envoie le message via LoRaWAN**

deviceState = DEVICE\_STATE\_CYCLE; **// Passe à l'état suivant**

break;

}

Ce qui se passe ici :

1. distanceCheck() récupère la distance mesurée par le LiDAR.
2. prepareTxFrame(appPort) prépare les données à envoyer.
3. LoRaWAN.send(); envoie la donnée via LoRaWAN.

Après l'envoi, l'appareil passe à l'état DEVICE\_STATE\_CYCLE pour attendre le prochain envoi.

### 5. Mise en veille et réveil :

L'appareil attend un certain temps avant d'envoyer la prochaine donnée :

case DEVICE\_STATE\_CYCLE:

{

distanceCheck(); **// Reprend une nouvelle mesure**

txDutyCycleTime = appTxDutyCycle + randr(-APP\_TX\_DUTYCYCLE\_RND, APP\_TX\_DUTYCYCLE\_RND); **// Définit le délai**

LoRaWAN.cycle(txDutyCycleTime); **// Programme l'envoi suivant**

deviceState = DEVICE\_STATE\_SLEEP;

break;

}

1. La prochaine mesure est prise.
2. Un délai aléatoire est ajouté pour éviter les collisions de paquets sur le réseau.
3. L’appareil passe en veille (DEVICE\_STATE\_SLEEP).

En mode DEVICE\_STATE\_SLEEP, l’appareil réduit sa consommation d'énergie jusqu'au prochain envoi :

case DEVICE\_STATE\_SLEEP:

{

LoRaWAN.sleep(loraWanClass);

break;

}

### 6. Résumé du fonctionnement LoRaWAN :

1. **Initialisation**
   * Configuration des clés et paramètres (OTAA/ABP, région, classe, etc.).
   * Initialisation de la communication LoRaWAN.
2. **Connexion au réseau**
   * Si OTAA est activé, l'appareil envoie une requête de join à la passerelle.
   * Si ABP est utilisé, l’appareil démarre directement.
3. **Envoi des données**
   * La distance mesurée est convertie en 2 octets.
   * Le message est envoyé à la passerelle LoRaWAN.
4. **Cycle de transmission**
   * Un délai est appliqué avant le prochain envoi.
   * L’appareil passe en veille pour économiser de l’énergie.

# Conclusion

Ce projet démontre l’intégration réussie d’un capteur de distance LiDAR Garmin avec une ESP32, permettant la mesure et l’affichage en temps réel des distances sur un écran OLED. Grâce à l’utilisation du protocole LoRa, ces données sont également transmises sans fil, garantissant une communication longue portée et à faible consommation.

L’architecture repose sur un busI2C pour la communication entre l’ESP32 et le LiDAR, ainsi qu’une interfacegraphiquesimplifiée sur OLED pour une lecture instantanée. Le choix de LoRaWAN assure une transmission efficace des mesures, ouvrant ainsi des perspectives pour des applications telles que la surveillanceindustrielle**,** lagestiondedistance entre véhicules ou la détection d’obstacles en environnements difficiles.

Ce projet met en avant une solution robuste et optimisée combinant capteurs de précision, affichage embarqué et communication IoT, offrant ainsi une base exploitable pour des développements futurs en télémétrie et en suivi d’objets à distance.