Projet Prototypage

Surveillance de niveau d'un liquide

Imane MOUMOUN

Table des matières

Introduction

Dans de nombreux domaines, qu'ils soient industriels, environnementaux ou liés à la recherche, la surveillance du niveau d'un liquide est une exigence technique fondamentale. Une mesure précise et fiable est indispensable pour garantir le bon fonctionnement des systèmes et éviter des situations critiques telles que le fonctionnement à vide d'une pompe, qui pourrait entraîner une surchauffe et une détérioration prématurée des équipements. Dans le secteur industriel, cette surveillance permet d'optimiser les processus de production, en assurant un approvisionnement constant et adapté aux besoins des machines et des chaînes de fabrication. Un contrôle précis contribue aussi à réduire les pertes et à améliorer la gestion des ressources, en limitant le gaspillage et en garantissant une meilleure efficacité énergétique. Dans les environnements où l'accès aux réservoirs est limité – comme dans les installations souterraines, les sites isolés ou les espaces confinés – le suivi du niveau des liquides devient un véritable défi. L'utilisation de capteurs adaptés et de technologies avancées permet d'assurer une surveillance à distance et d'éviter les interventions humaines fréquentes, souvent coûteuses et complexes.

C'est dans cette perspective que s'inscrit notre projet : concevoir un système de mesure de niveau de liquide fiable, robuste et adaptable, reposant sur un principe capacitif. Celui-ci repose sur l'utilisation de deux électrodes recouvertes d'un isolant, immergées dans le liquide à surveiller. Cette couche isolante a un double rôle : elle évite les réactions chimiques potentielles entre les électrodes et le liquide, et prévient les courts-circuits dans le cas de liquides conducteurs. La capacité mesurée entre les électrodes varie en fonction du niveau atteint par le liquide, permettant ainsi une estimation avec une bonne stabilité.

Dans notre cas, les expérimentations seront réalisées sur de l'eau, un liquide courant et pertinent pour de nombreuses applications. Néanmoins, le système est pensé pour être généralisable à d'autres liquides, en permettant une phase de calibrage simple dès que le fluide change.

Ce double objectif, fonctionnel (mesure de niveau) et analytique (comportement diélectrique), guide toute la conception du dispositif. Le système devra produire un signal mesurable, stable, et sensible aux variations de niveau.

Notre projet se structure en trois étapes principales : la conception d'un capteur capacitif, le développement d'un conditionneur chargé d'extraire la valeur de la capacité et de la

transformer en grandeur exploitable, en l'occurrence la fréquence, puis l'intégration complète du système suivie de sa phase de test. Les données mesurées seront affichées sur un écran.

Notre démarche s'appuie sur une approche expérimentale, complétée par des phases de modélisation numérique et d'analyse théorique. La conception du système s'est fondée sur l'étude de documents scientifiques, l'analyse de fiches techniques de composants électroniques, ainsi que sur les conseils et l'accompagnement de nos encadrants.

Pour mener à bien ce projet, nous avons utilisé plusieurs outils logiciels :

- WaveForms pour la visualisation des signaux de sortie du circuit conditionneur
- COMSOL pour la simulation du comportement physique du capteur capacitif.
- Autodesk Inventor pour la modélisation 3D.
- STM32CubeIDE pour la programmation du microcontrôleur.

Enfin, notre travail s'est appuyé sur les enseignements théoriques dispensés en première année du cursus ISMIN, notamment les cours de Physique des Composants Passifs et de Prototypage, qui ont fourni les bases nécessaires à la conception et à la réalisation du système.

1. Vue générale du système

La mesure du niveau d'un liquide peut être réalisée par divers moyens. Dans notre projet, nous avons choisi une approche basée sur un capteur capacitif, pour sa simplicité, sa robustesse et sa capacité d'adaptation à différents types de liquides.

Le principe repose sur la variation de la capacité électrique entre deux électrodes, en fonction de la permittivité diélectrique du milieu dans lequel elles sont partiellement immergées. Cette permittivité varie avec la nature du liquide et le niveau atteint. Ainsi, lorsque le niveau change, la capacité du capteur évolue de manière mesurable.

Plutôt que de supposer une connaissance théorique de la constante diélectrique du liquide (ce qui limiterait la précision dans des conditions réelles), nous avons choisi une approche expérimentale. Une phase initiale de calibration consiste à remplir le réservoir jusqu'à un niveau connu pour déterminer empiriquement la constante diélectrique du liquide utilisé. Cette méthode garantit une mesure plus fiable par la suite et permet également de suivre l'évolution du comportement diélectrique du fluide.

La capacité du capteur est ensuite convertie en signal électrique via un circuit conditionneur, qui en extrait une grandeur exploitable, typiquement une fréquence. Le signal est ensuite traité par un microcontrôleur, qui calcule la hauteur du liquide et l'affiche en temps réel sur un écran OLED. Par ailleurs, les données recueillies par le capteur peuvent également être transmises via un module Bluetooth.

La figure ci-dessous illustre le fonctionnement global de notre système, depuis la détection capacitive, en passant par la transduction et le traitement du signal, jusqu'à l'affichage final.

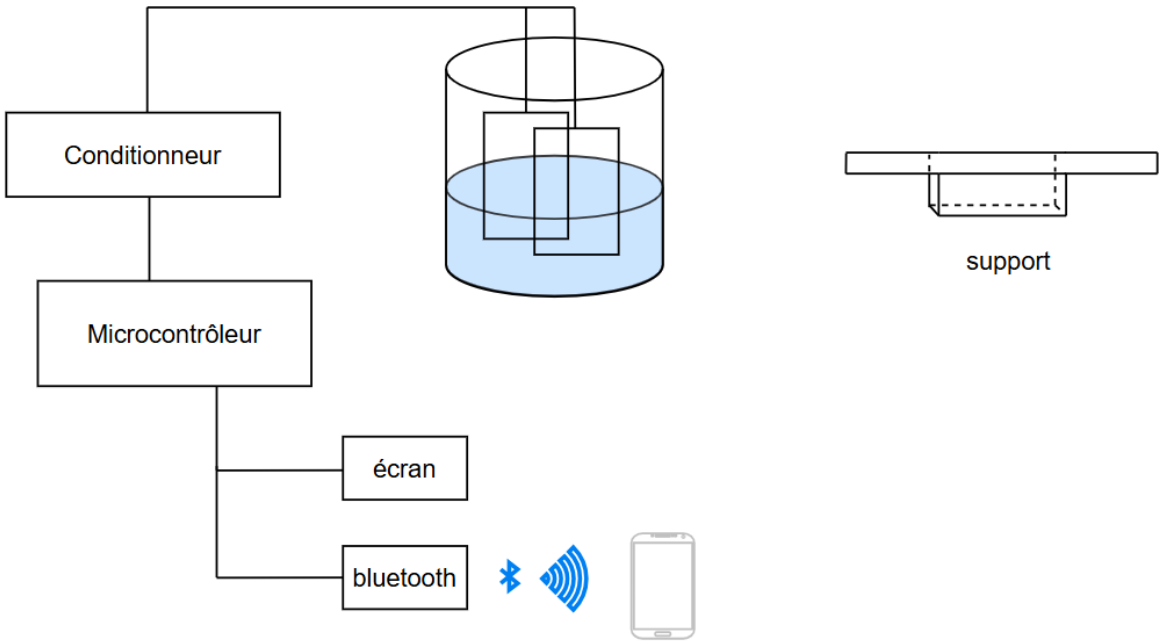

Figure 1 : Schéma global du système de surveillance de niveau

2. Conception du capteur capacitif

L'idée de notre projet Prototypage est de travailler en utilisant un capteur capacitif. Nous allons ainsi détailler le fonctionnement de ce type de capteur ainsi que la conception du conditionneur utilisé pour obtenir un résultat exploitable sur un microcontrôleur à partir de la variation d'une capacité.

2.1. Aspects physiques mis en jeu

Dans le cadre de cette étude, nous nous intéressons à l'utilisation d'un condensateur plan comme capteur de niveau d'eau. Le principe repose sur la variation de la capacité en fonction de la hauteur d'eau entre les armatures, compte tenu des différentes permittivités des milieux

traversés (air, eau, isolant). Une modélisation rigoureuse permet d'exprimer la capacité totale du système en fonction de cette hauteur [1].

Rappel : Capacité d'un condensateur plan

Un condensateur plan constitué de deux armatures parallèles de surface S, séparées par une distance d, remplies d'un milieu homogène de permittivité relative ε_r, possède une capacité : $C = \varepsilon_0 \varepsilon_r \frac{S}{d}$ où ε_0 est la permittivité du vide ($\simeq 8.85 \times 10^{-12} F/m$).

Remarque : on néglige les effets de bords pour $S \gg d^2$.

La capacité dépend donc :

- De la géométrie du condensateur (S et d) ;
- De la nature du milieu diélectrique (ε_r).

Capacité entre milieux de permittivités différentes : Influence de la hauteur d'eau

Dans le cas où les armatures sont partiellement plongées dans l'eau (de permittivité $\varepsilon_{r,eau} \simeq 80$) et partiellement dans l'air ($\varepsilon_{r,air} \simeq 80$), la capacité totale est équivalente à deux condensateurs en parallèle (figure 2):

$$C = C_{air} + C_{eau} = \varepsilon_0 \frac{S/H}{d} [\varepsilon_{r,air} \times (H - h) + \varepsilon_{r,eau} \times h]$$

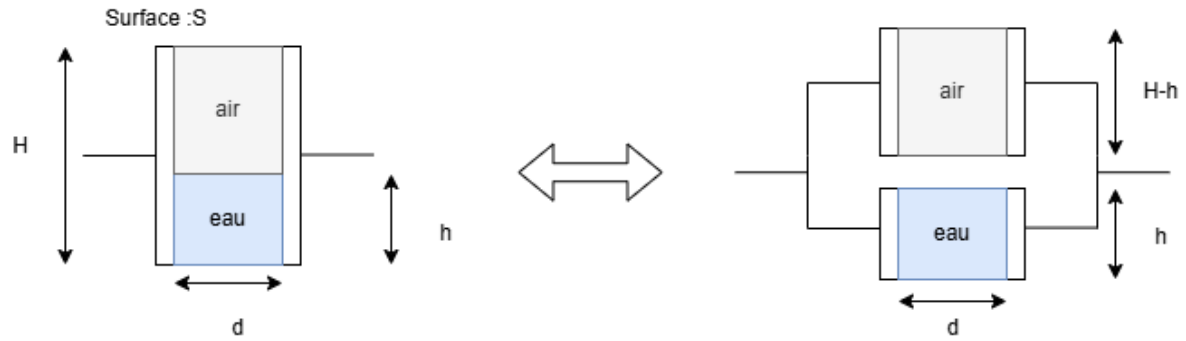

Figure 2 :Association parallèle

Nécessité d'une couche isolante et association en série

Dans le cas d'un capteur de niveau d'eau, il est essentiel d'ajouter une couche isolante entre les armatures métalliques et le milieu aqueux. En effet, l'eau présente dans les environnements

réels n'est jamais parfaitement pure et se comporte comme un conducteur électrique en raison des ions dissous (sels minéraux, impuretés, etc.).

Si les armatures étaient en contact direct avec cette eau conductrice, cela entraînerait un court-circuit entre les plaques, une altération de la mesure de capacité voire une dégradation du dispositif électronique.

Pour éviter cela, chaque armature est donc recouverte d'une fine couche d'isolant, ce qui permet de maintenir l'intégrité électrique du système tout en autorisant la détection capacitive via les variations de permittivité du milieu environnant (air ou eau).

Du point de vue physique, cette configuration revient à considérer une association en série de plusieurs couches diélectriques de permittivités différentes entre les armatures (figure 3). La capacité résultante s'exprime alors comme :

$$\frac{1}{C} = \frac{1}{C_A} + \frac{1}{C_B} = \frac{1}{\varepsilon_0 \,.\, S}\left(\frac{d_A}{\varepsilon_{r,A}} + \frac{d_B}{\varepsilon_{r,B}}\right)$$

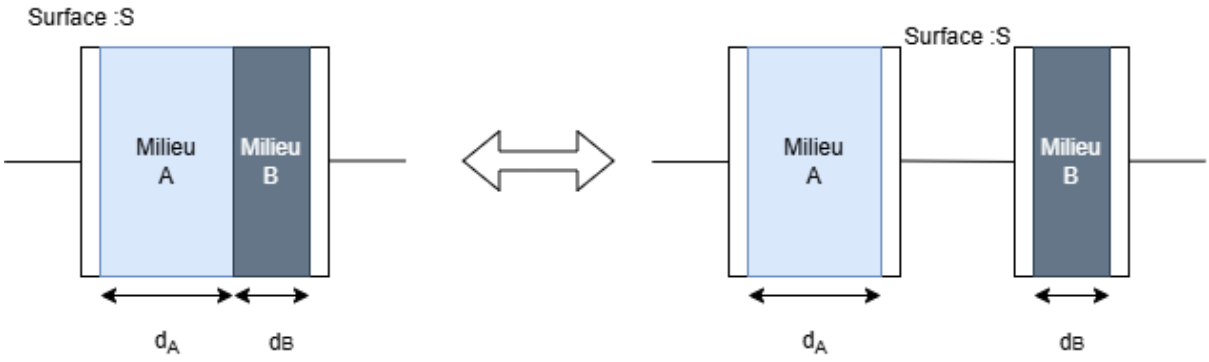

Figure 3 : Association série

Application au capteur de niveau d'eau

Dans notre contexte d'étude, le volume situé entre les deux armatures du condensateur est constitué de deux régions superposées verticalement : une zone supérieure non immergée, remplie d'air et une zone inférieure immergée dans l'eau.

Chaque portion est modélisée comme une association en série de trois couches diélectriques : une couche isolante déposée sur la première armature, le milieu (air ou eau), puis la couche isolante sur la seconde armature (figure 4).

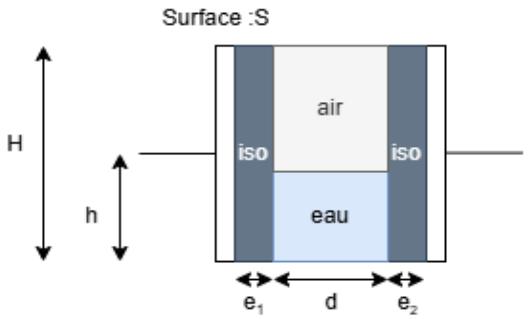

Figure 4 : Modélisation du capteur

La capacité totale du système est alors obtenue comme la somme en parallèle de ces deux capacités :

$$C = C_{air} + C_{eau}$$

avec $\frac{1}{C_{air}} = \frac{1}{C_{iso1}} + \frac{1}{C_{air}} + \frac{1}{C_{iso2}} = \frac{1}{\varepsilon_0 \cdot \frac{S}{H}.(H-h)} \left(\frac{e_1+e_2}{\varepsilon_{r,iso}} + \frac{d}{\varepsilon_{r,air}}\right)$

et $\frac{1}{C_{eau}} = \frac{1}{C\prime_{iso1}} + \frac{1}{C_{air}} + \frac{1}{C\prime_{iso2}} = \frac{1}{\varepsilon_0 \cdot \frac{S}{H}.h} \left(\frac{e_1+e_2}{\varepsilon_{r,iso}} + \frac{d}{\varepsilon_{r,eau}}\right)$

ce qui donne : $C = \varepsilon_0 . \frac{S(H-h)}{H} . \frac{1}{\left(\frac{e_1+e_2}{\varepsilon_{r,iso}} + \frac{d}{\varepsilon_{r,air}}\right)} + \varepsilon_0 . \frac{S.h}{H} . \frac{1}{\left(\frac{e_1+e_2}{\varepsilon_{r,iso}} + \frac{d}{\varepsilon_{r,eau}}\right)}$

$$C = \varepsilon_0 . \frac{S}{H} \left(\frac{1}{\left(\frac{e_1+e_2}{\varepsilon_{r,iso}} + \frac{d}{\varepsilon_{r,eau}}\right)} - \frac{1}{\left(\frac{e_1+e_2}{\varepsilon_{r,iso}} + \frac{d}{\varepsilon_{r,air}}\right)}\right).h + \frac{\varepsilon_0 . S}{\left(\frac{e_1+e_2}{\varepsilon_{r,iso}} + \frac{d}{\varepsilon_{r,air}}\right)}$$

Dès lors, cette étude met en évidence que la capacité totale varie linéairement avec la hauteur d'eau entre les armatures, même en présence de couches isolantes. Cela permet d'utiliser un tel système comme capteur de niveau, à condition de tenir compte des permittivités des matériaux en jeu. La modélisation en série-parallèle permet une description précise du comportement du capteur et ouvre la voie à un étalonnage fiable.

$$C = \alpha . h + C_{air,totale}$$

où :

- α la pente est qui dépend de la permittivité du liquide,
- $C_{air,totale}$ est la capacité mesurée lorsque le capteur est totalement à l'air.

2.2. Simulation COMSOL de la capacité

Afin de valider notre modélisation théorique et d'estimer l'ordre de grandeur des capacités attendues, nous avons utilisé le logiciel COMSOL Multiphysics pour simuler le comportement du capteur [2]. L'objectif principal de cette simulation est de vérifier l'évolution de la capacité en fonction de la hauteur d'eau entre les électrodes, et notamment de confirmer la linéarité théorique de cette relation.

Pour cela, nous avons réalisé une étude paramétrique dans laquelle nous avons fait varier progressivement la hauteur du liquide, tout en observant les valeurs de capacité obtenues à chaque étape. Cette démarche nous a permis non seulement de valider expérimentalement notre modèle analytique, mais aussi d'affiner les paramètres géométriques de notre capteur, notamment la surface des électrodes et l'espacement entre les armatures, afin d'en améliorer la sensibilité et la précision.

La simulation a été réalisée avec les paramètres suivants :

Surface des armatures : $S = 160\ cm^2$

Hauteur totale : $H = 16\ cm$

Épaisseurs des couches isolantes : $e_1 = e_2 = 0.2\ mm$

Distance entre les armatures : $d = 1\ cm$

Permittivité relative de l'isolant : $\varepsilon_{r,iso} = 2.7$

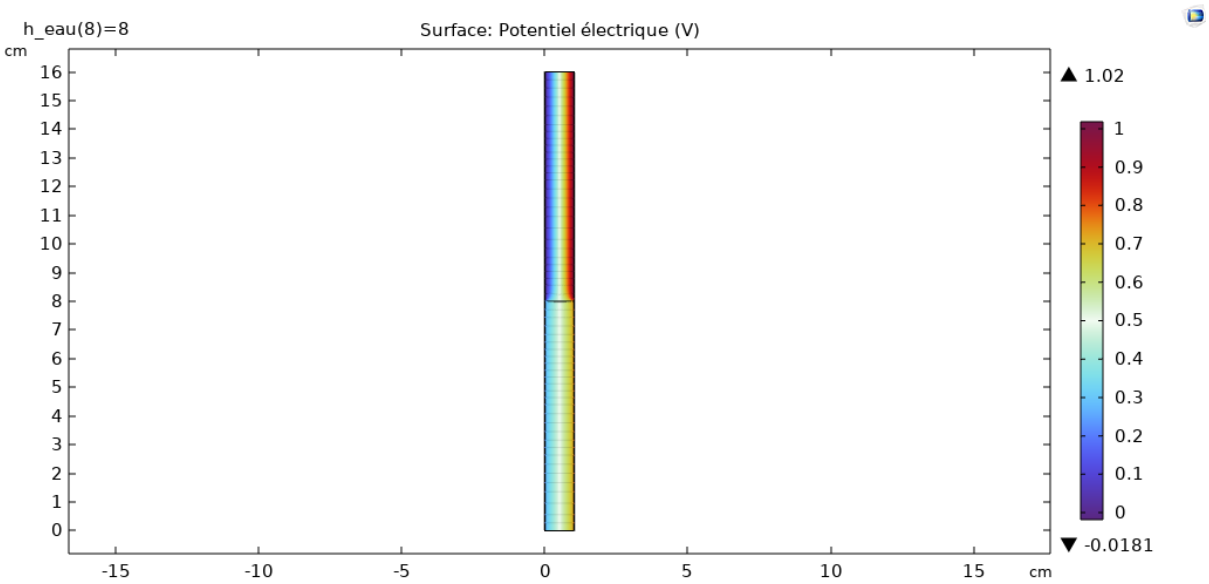

Figure 5 : Simulation 2D sur COMSOL du capteur

La figure ci-dessus représente la distribution du potentiel électrique dans le condensateur pour une hauteur d'eau de 8 cm. On distingue deux zones de comportement distinct :

- La partie inférieure (0 à 8 cm) correspond à la région immergée dans l'eau. Elle présente un gradient de potentiel plus faible, ce qui est cohérent avec la forte permittivité diélectrique de l'eau ($\varepsilon_{r,eau} \approx 80$). En effet, plus la permittivité est élevée, plus le champ électrique (et donc le gradient de potentiel) est réduit à capacité équivalente.

- La partie supérieure (8 à 16 cm) est occupée par l'air, dont la permittivité est beaucoup plus faible ($\varepsilon_{r,air} \approx 1$). On y observe un gradient de potentiel plus prononcé, ce qui confirme que le champ électrique est plus intense dans les milieux faiblement polarisables.

Ces résultats confirment le comportement théorique attendu : le potentiel est inversement proportionnel à la permittivité du milieu traversé. Ainsi, à géométrie constante, plus le liquide monte dans le capteur, plus la proportion du champ traversant un milieu à forte permittivité augmente, ce qui se traduit par une modification mesurable de la capacité totale du système.

Les valeurs de la capacité en fonction de la hauteur d'eau ont été extraites de l'étude paramétrique réalisée sous COMSOL, puis exportées dans un tableur. La courbe obtenue (figure 6) montre une variation linéaire de la capacité en fonction de la hauteur d'immersion, ce qui est en accord avec le modèle théorique développé précédemment.

Une application numérique de la formule analytique permet de valider la cohérence des résultats de simulation, confirmant que la capacité dépend linéairement de la hauteur d'eau, conformément à l'expression démontrée précédemment.

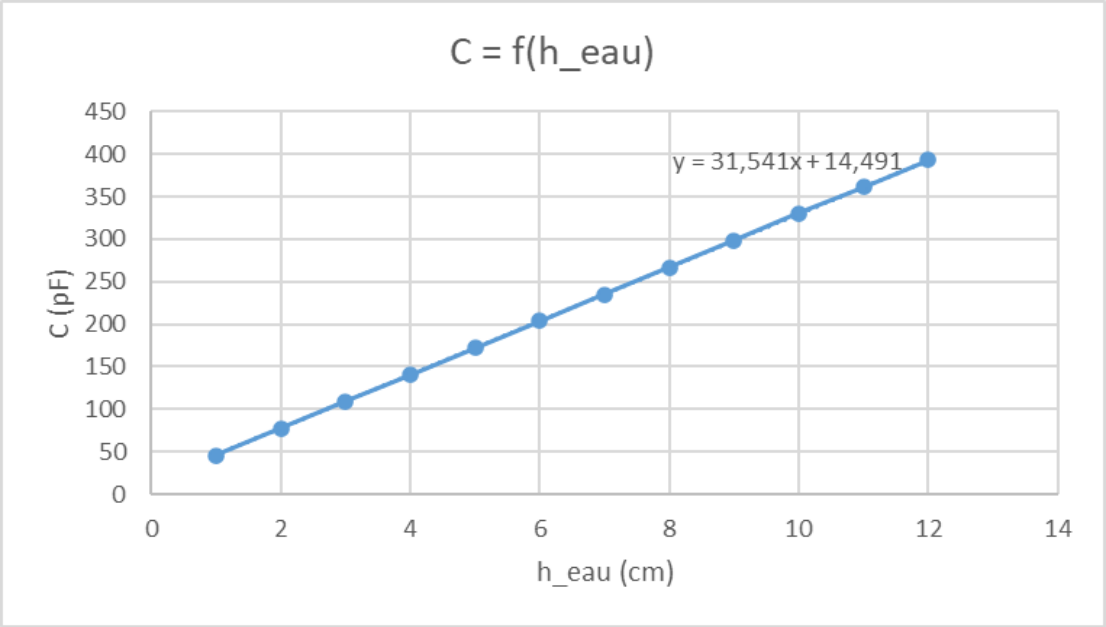

Figure 6 : Variation théorique de la capacité du capteur en fonction de la hauteur de l'eau

3. Conditionneur

La capacité mesurée par notre capteur est de l'ordre du picofarad. Ce faible ordre de grandeur nécessite une méthode de lecture particulièrement sensible et robuste, capable de fournir une mesure fiable tout en minimisant les perturbations.

Pour cela, nous avons opté pour un conditionneur à courant constant (figure 7), basé sur le principe de mesure du temps de charge d'un condensateur. L'idée est d'associer la capacité inconnue à un oscillateur dont la fréquence d'oscillation dépend directement du temps de charge de la capacité sous un courant constant. En mesurant cette fréquence, il devient possible de remonter à la valeur de la capacité, grâce à une relation linéaire entre les deux grandeurs. Cette approche présente également l'avantage de réduire l'influence des bruits de tension et des interférences parasites.

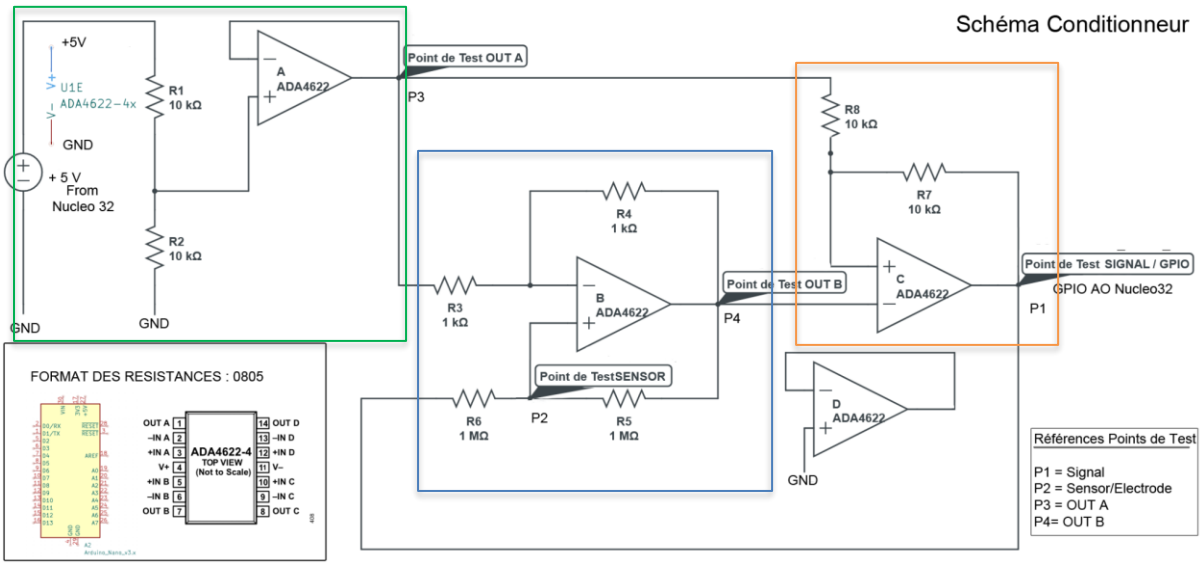

Figure 7 : Schéma du conditionneur

Le conditionneur que nous avons conçu repose sur trois blocs fonctionnels principaux :

- Un générateur de tension de référence (figure 7 bloc vert) : il permet d'établir une demi-alimentation (2,5 V) via un pont diviseur, afin de centrer le signal de l'oscillateur entre 0 V et 5 V, soit les niveaux d'alimentation standards du microcontrôleur.
- Un convertisseur tension-courant (figure 7 bloc bleu) : ce circuit applique un courant constant à la capacité (figure 8 courbe bleue), condition indispensable pour garantir une relation linéaire entre la capacité et la fréquence d'oscillation. Il est commandé par une tension continue.

- Un comparateur à hystérésis (figure 7 bloc orange) : il fonctionne comme un oscillateur, en créant une boucle de rétroaction entre sa sortie et le convertisseur tension-courant (figure 8 courbe orange). Ce montage produit une oscillation carrée dont la fréquence est donnée par la formule : $f = \frac{1}{RC}$, où R est une résistance fixe du circuit, et C la capacité à mesurer et qui sera branchée sur TestSENSOR.

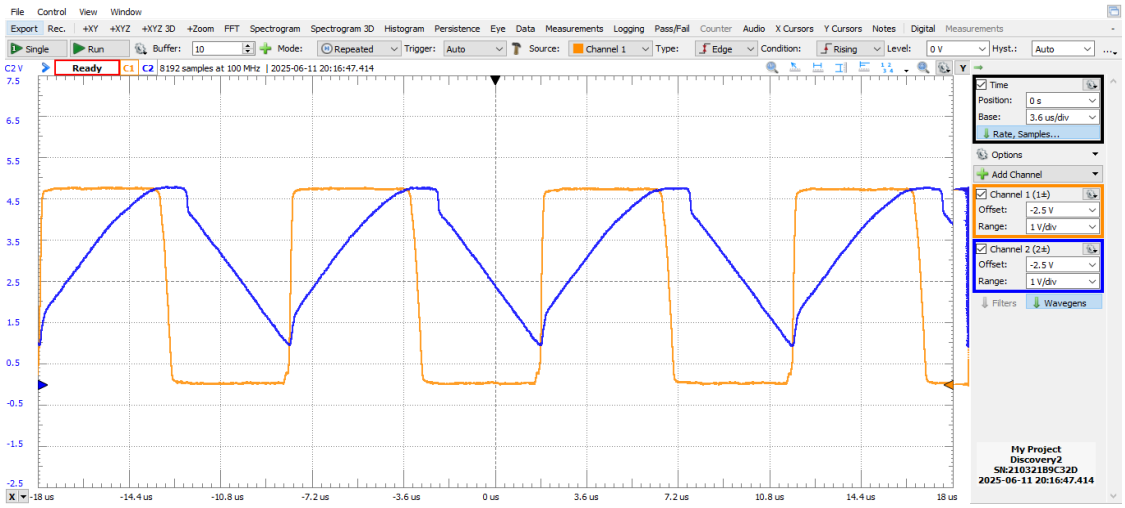

Figure 8 : Visualisation des signaux de sortie sur WaveForms

Afin de caractériser le comportement du conditionneur, nous avons utilisé plusieurs condensateurs de capacité connue. Les mesures obtenues ont permis de tracer une droite affine (figure 9) représentant la relation entre la fréquence d'oscillation et la capacité.

L'extrapolation de cette courbe par régression linéaire met en évidence la présence d'une capacité parasite de 19 pF lorsqu'il n'y a pas d'oscillations dans le conditionneur. Il s'agit de la capacité que nous allons chercher à négliger lors du dimensionnement des électrodes.

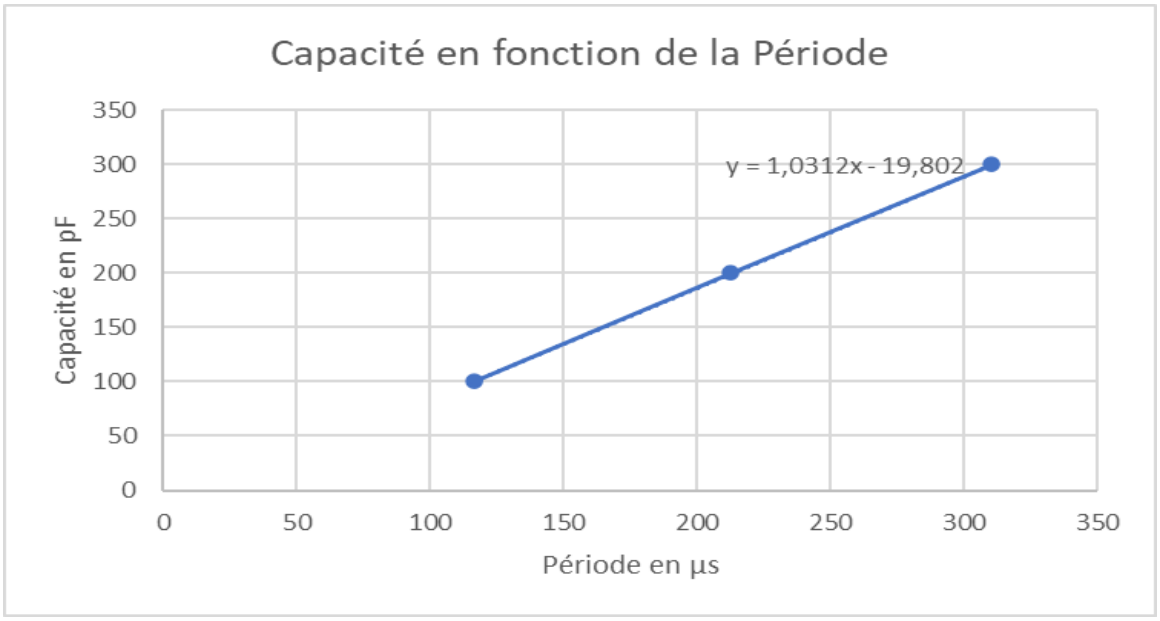

Figure 9 : Mise en évidence de la relation entre capacité et période

4. Assemblage du prototype

Dans le cadre de la validation de notre projet, nous avons réalisé un prototype expérimental. Il se compose d'un bécher rempli d'eau, dans lequel sont plongées deux électrodes isolées (figure 10), fixées sur un support imprimé en 3D. Ce dispositif est relié à notre circuit conditionneur, au microcontrôleur STM32F301K8 [3], ainsi qu'à un écran OLED et à un module Bluetooth, tous deux montés sur une breadboard (figure 11).

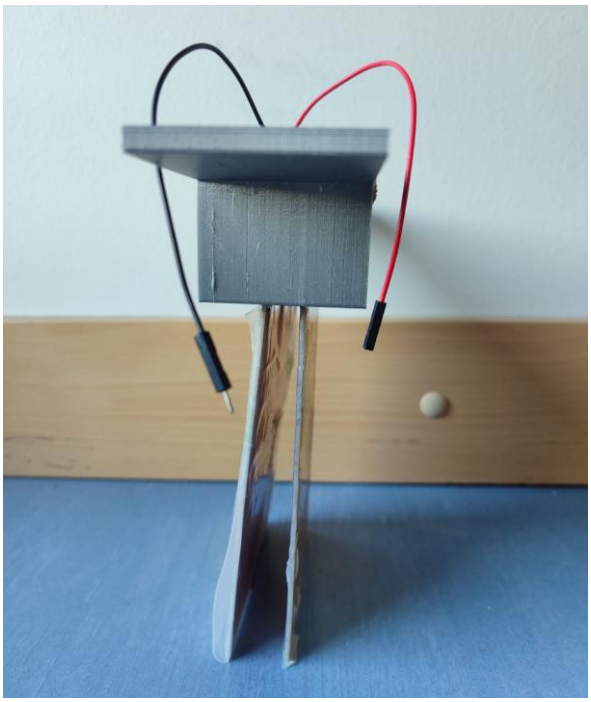

Figure 10 : Support et électrodes

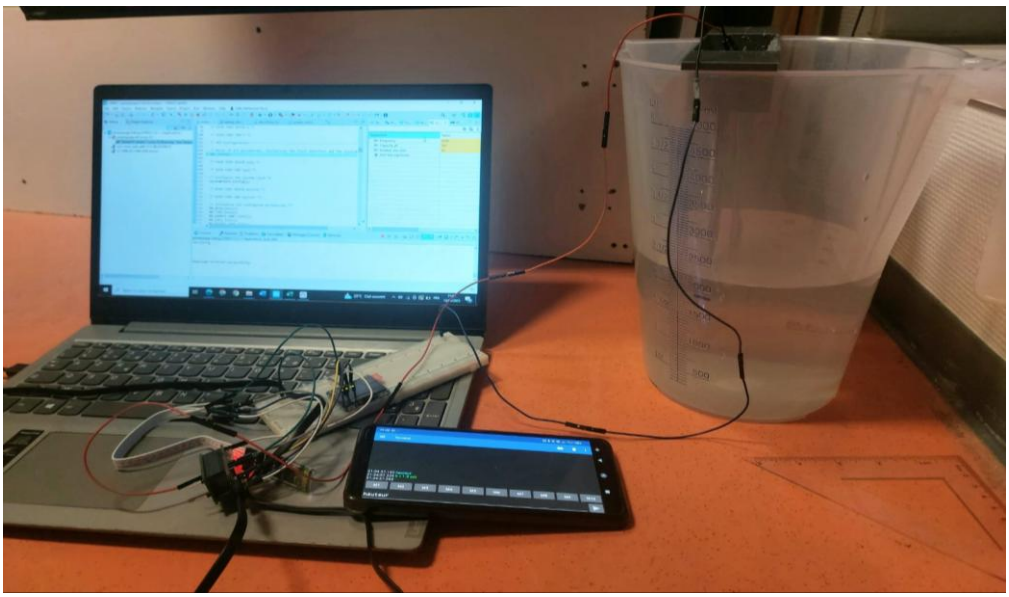

Figure 11 : Vue globale du prototype expérimental

L’utilisateur interagit avec le système via son téléphone, qui sert d’interface de commande. Lors de la première utilisation, il peut envoyer la commande calibre afin de lancer la phase de calibration. Ensuite, la commande hauteur permet d’obtenir la hauteur d’eau dans le bécher. Si cette hauteur devient inférieure à un seuil prédéfini, le système envoie automatiquement une alerte à l’utilisateur par Bluetooth (figure 12). L’affichage est en permanence assuré par l’écran OLED (figure 13).

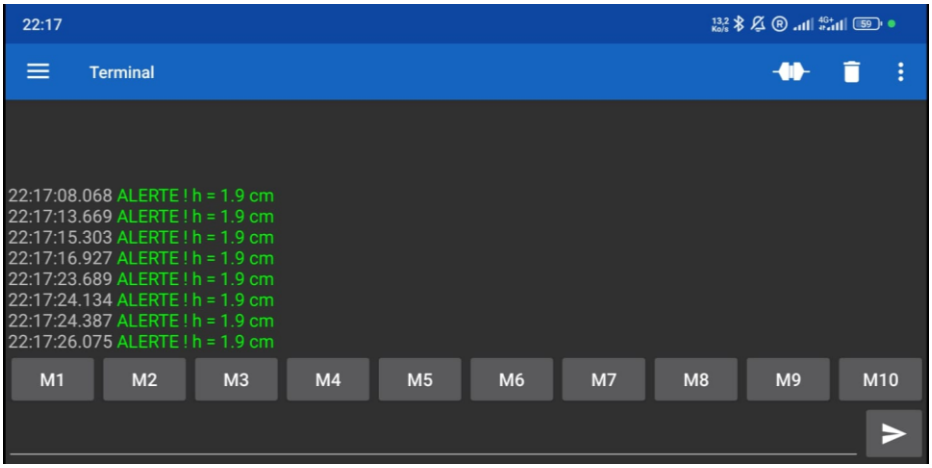

Figure 12 : Déclenchement alerte via Bluetooth

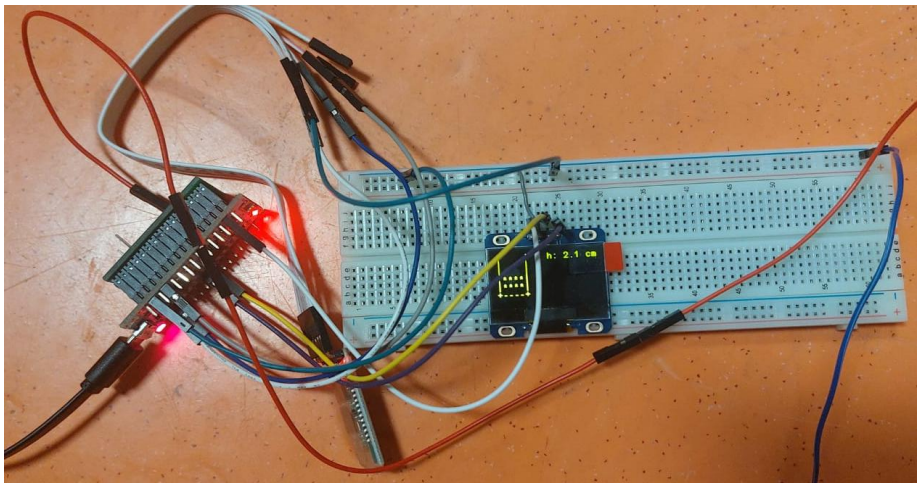

Figure 13 : Affichage niveau eau par écran OLED

Dans le cadre standard de notre projet, où l’eau constitue le sujet d’étude, les calculs théoriques permettent d’établir une équation reliant la capacité mesurée à la hauteur du liquide. Toutefois, certains paramètres, notamment l’épaisseur de la couche isolante recouvrant chaque électrode, demeurent difficiles à quantifier avec précision.

Pour pallier cette incertitude, une première phase de calibration a été menée. Elle a consisté à mesurer la capacité pour différentes hauteurs d'eau connues, ce qui nous a permis de tracer la courbe caractéristique du capteur. Cette courbe (figure 14), de nature linéaire, a confirmé le bon fonctionnement du prototype.

En intégrant cette relation empirique au système, il devient alors possible de déduire la hauteur de l'eau à partir de la capacité mesurée.

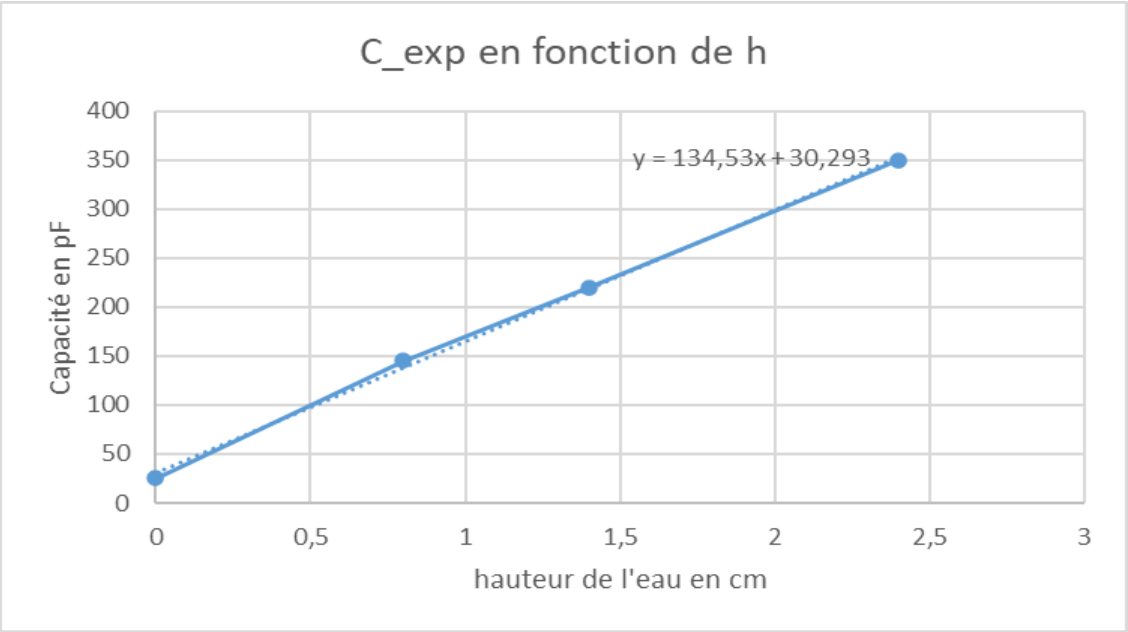

Figure 14 : Variation expérimentale de la capacité du capteur en fonction de la hauteur de l'eau

5. Algorithme de fonctionnement

Le code STM32 est conçu pour gérer les différents périphériques constituant notre projet, à savoir l'écran OLED et le module Bluetooth, tout en assurant les calculs nécessaires au bon fonctionnement du système.

Pour mesurer la capacité du circuit, on détermine la période du signal en sortie du conditionneur. À cet effet, le Timer2 est configuré en mode *Input Capture*. Lorsqu'un front montant est détecté, la valeur du compteur du Timer2 est automatiquement enregistrée (IC1). Lorsqu'un second front est détecté, une nouvelle capture est effectuée (IC2), ce qui permet de calculer la période du signal mesuré. Cette période correspond à la différence entre les deux valeurs capturées, divisée par la fréquence du timer, soit :

$$Période = \frac{IC2 - IC1}{HAL\_RCC\_GetPCLK1Freq()}$$

Une fois la capacité mesurée, la hauteur du liquide est calculée à partir d'une formule déterminée expérimentalement. Le code de base est conçu pour fonctionner avec de l'eau, mais une phase de calibration est possible via l'envoi de la commande calibre par Bluetooth, afin d'ajuster la pente de l'équation reliant la capacité et la hauteur.

Le module Bluetooth permet également d'envoyer à l'utilisateur la hauteur du liquide sur requête hauteur, ou d'émettre automatiquement une alerte si la hauteur devient inférieure à un seuil critique. L'écran OLED permet d'afficher en temps réel le niveau du liquide présent dans le réservoir.

L'algorigramme ci-dessous décrit la logique de fonctionnement du code du microcontrôleur.

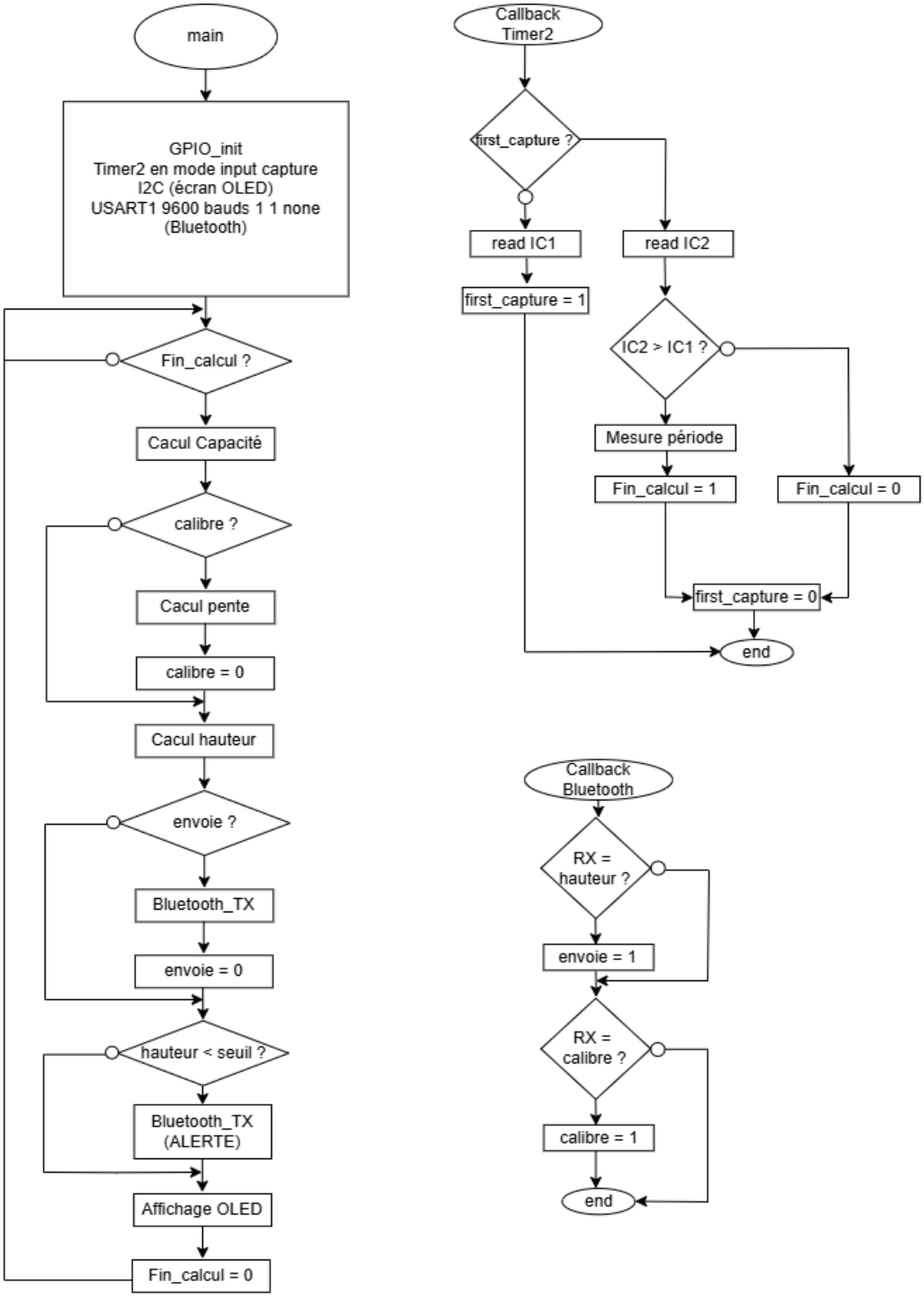

Figure 15 : Algorigramme du code

Le code source complet est présenté en annexe.

Synthèse et ouverture

Ce travail a abouti à la réalisation d'un dispositif de mesure de niveau de liquide utilisant un capteur capacitif, avec une démarche axée sur l'expérimentation pour maximiser la précision. Plutôt que de nous appuyer sur une valeur théorique de la constante diélectrique, nous avons opté pour une calibration empirique basée sur des hauteurs connues de liquide. Cette méthode a permis d'établir une relation fiable entre la capacité mesurée et la hauteur, confirmée par la linéarité observée des résultats expérimentaux. Le bon fonctionnement du capteur ainsi que la cohérence du modèle ont ainsi été validés.

Plusieurs difficultés ont été rencontrées. L'impression 3D du boîtier, initialement prévue pour isoler les électrodes, s'est révélée inadaptée en raison de dimensions trop réduites. Par ailleurs, l'isolation électrique des électrodes a constitué un autre défi : malgré l'utilisation de vernis et de ruban adhésif, l'isolation des bords n'a pas pu être assurée de manière satisfaisante, ce qui a pu affecter la précision des mesures.

Malgré ces obstacles, le prototype final est pleinement fonctionnel et remplit les objectifs fixés. Il offre une solution simple, interactive et efficace pour le suivi du niveau d'eau dans un récipient.

Plusieurs perspectives d'amélioration peuvent être envisagées pour perfectionner ce système. D'un point de vue matériel, le support des électrodes pourrait être repensé afin d'être plus facilement imprimable en 3D, avec une meilleure stabilité mécanique. L'isolation des électrodes pourrait quant à elle être optimisée à l'aide de matériaux plus adaptés, comme des gaines thermo-rétractables ou un revêtement isolant uniforme appliqué en usine. Sur le plan électronique, remplacer la breadboard par un circuit imprimé (PCB) offrirait une meilleure fiabilité des connexions. Enfin, le système pourrait évoluer vers une solution entièrement autonome, intégrant par exemple une alimentation sur batterie, un module de transmission à distance longue portée (LoRa ou Wi-Fi), ou encore des fonctions de journalisation des mesures.

Au-delà de ces améliorations techniques, ce dispositif pourrait être intégré dans des environnements variés : la gestion des niveaux d'eau dans des réservoirs domestiques ou agricoles, la surveillance automatisée de citernes, ou encore le contrôle de l'humidité dans des serres ou systèmes d'irrigation intelligents. Dans ces contextes, l'intégration de l'intelligence artificielle constituerait une évolution pertinente. L'IA pourrait, par exemple, analyser les

données historiques de hauteur d'eau pour anticiper les besoins en remplissage, détecter des anomalies de consommation ou optimiser les cycles d'arrosage en fonction de prévisions météorologiques et de l'usage.

Ainsi, notre prototype pose les bases d'un système intelligent et adaptable, à fort potentiel pour les applications liées à l'environnement, à la domotique et à l'agriculture connectée.

Références

[1] 5.14: Mixed Dielectrics - Physics LibreTexts

[2] M.Saadaoui – Physique de Composants Passifs – Introduction à COMSOL MULTIPHYSICS

[3] STM32 Nucleo-32 boards (MB1180) - User manual

Annexes

```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2025 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "stdio.h"
#include "string.h"
#include "ssd1306.h"
#include "fonts.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define  R 1  //MHz
#define Cparasite 19 //pF
#define Cairtotale 30 //Capacité totale sans liquide en pF
#define h_calibre 2 //cm
#define SEUIL 20 //mm
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
I2C_HandleTypeDef hi2c1;

TIM_HandleTypeDef htim2;

UART_HandleTypeDef huart1;
UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
uint8_t Is_First_Captured = 0;
uint32_t IC_Value1 = 0;
uint32_t IC_Value2 = 0;
uint32_t Difference = 0;
uint32_t Frequency = 0;
uint8_t Calcul_Fin = 0;
uint32_t Period_us = 0;
int Capacity_pf = 0;
char msg[50];
```

```c
uint8_t calibre = 0;
uint32_t pente_eps = 134; //pente C = f(h) expérimentale avec de l'eau
int hauteur_eau_mm = 0;
//char oled_buffer[50];
uint8_t rx_buffer[10];
uint8_t envoie = 0;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_Init(void);
static void MX_USART2_UART_Init(void);
static  void  MX_I2C1_Init(void);
static void MX_USART1_UART_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
void afficher_reservoir_oled(uint8_t hauteur_eau_mm)
{
    const uint8_t max_hauteur = 5; // nombre de lignes pour la hauteur max
    char ligne[10];

    SSD1306_Clear();
    SSD1306_GotoXY(0, 0);

    for (int i = max_hauteur; i > 0; i--)
    {
        if (i <= hauteur_eau_mm/10)
            strcpy(ligne, "|****|");
        else
            strcpy(ligne, "|    |");

        SSD1306_Puts(ligne, Font_7x10, 1);
        SSD1306_GotoXY(0, (max_hauteur - i + 1) * 10);  // espacement vertical (10 px/ligne)
    }

    // Dessiner le fond
    SSD1306_Puts("+----+", Font_7x10, 1);

    // Taux de remplissage
    char hauteur[30];
    sprintf(hauteur, "h: %d.%d cm", hauteur_eau_mm/10, hauteur_eau_mm%10);
    uint8_t text_width = strlen(hauteur) * 7;
    SSD1306_GotoXY(128 - text_width, 0);  // aligné à droite
    SSD1306_Puts(hauteur, Font_7x10, 1);

    SSD1306_UpdateScreen();
}

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */
```

```
/* MCU Configuration--------------------------------------------------------*/

/* Reset of all peripherals, Initializes the Flash interface and the Systick. */
HAL_Init();

/* USER CODE BEGIN Init */

/* USER CODE END Init */

/* Configure the system clock */
SystemClock_Config();

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_TIM2_Init();
MX_USART2_UART_Init();
MX_I2C1_Init();
MX_USART1_UART_Init();
/* USER CODE BEGIN 2 */
HAL_TIM_IC_Start_IT(&htim2,TIM_CHANNEL_1);

SSD1306_Init();
SSD1306_Clear();
SSD1306_GotoXY(32, 16);
SSD1306_Puts("Init OLED", Font_7x10, 1);
SSD1306_UpdateScreen();
HAL_Delay(1000);
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
 if (Calcul_Fin)
 {
           Capacity_pf = (1/R)*Period_us - Cparasite;
         sprintf(msg, "Capacity = %d pf\n\r", Capacity_pf);
         HAL_UART_Transmit(&huart2, (uint8_t*)msg, sizeof(msg), 200);

         if (Capacity_pf >= Cairtotale)
         {
          HAL_UART_Receive_IT(&huart1, rx_buffer, sizeof(rx_buffer));

          if (calibre)
        {
           pente_eps  =  (Capacity_pf-Cairtotale)/h_calibre;
           sprintf(msg, "pente = %ld \n\r", pente_eps);
           HAL_UART_Transmit(&huart1, (uint8_t*)msg, strlen(msg), 200);
           calibre = 0;
        }
       hauteur_eau_mm = 10*(Capacity_pf - Cairtotale)/pente_eps;


          HAL_UART_Receive_IT(&huart1, rx_buffer, sizeof(rx_buffer));
          if (envoie)
           {
           sprintf(msg, "h = %d.%d cm\n\r", hauteur_eau_mm/10, hauteur_eau_mm%10);
           HAL_UART_Transmit(&huart2, (uint8_t*)msg, sizeof(msg), 200);
           HAL_UART_Transmit(&huart1, (uint8_t*)msg, strlen(msg), 200);
           envoie = 0;
           }
          else if (hauteur_eau_mm < SEUIL)
```

```
            {
            sprintf(msg, "ALERTE ! h = %d.%d cm\n", hauteur_eau_mm/10, hauteur_eau_mm%10);
            HAL_UART_Transmit(&huart2, (uint8_t*)msg, sizeof(msg), 200);
            HAL_UART_Transmit(&huart1, (uint8_t*)msg, strlen(msg), 200);
            }

           afficher_reservoir_oled(hauteur_eau_mm);
//           sprintf(oled_buffer, "h = %d.%d cm", hauteur_eau_mm/10, hauteur_eau_mm%10);
//           SSD1306_Clear();
//           SSD1306_GotoXY(0, 0);
//           SSD1306_Puts(oled_buffer, Font_7x10, 1);
//           SSD1306_UpdateScreen();
         }

         Calcul_Fin = 0;
   }

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL8;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
   RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                               |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_USART1|RCC_PERIPHCLK_I2C1;
  PeriphClkInit.Usart1ClockSelection = RCC_USART1CLKSOURCE_SYSCLK;
  PeriphClkInit.I2c1ClockSelection = RCC_I2C1CLKSOURCE_HSI;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
```

```
    Error_Handler();
  }
}

/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.Timing = 0x00201D2B;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Analogue filter
  */
  if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Digital filter
  */
  if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */

}

/**
  * @brief TIM2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM2_Init(void)
{

  /* USER CODE BEGIN TIM2_Init 0 */

  /* USER CODE END TIM2_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
```

```c
  TIM_IC_InitTypeDef sConfigIC = {0};

  /* USER CODE BEGIN TIM2_Init 1 */

  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 0;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 4294967295;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_IC_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
  sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
  sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
  sConfigIC.ICFilter = 0;
  if (HAL_TIM_IC_ConfigChannel(&htim2, &sConfigIC, TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

  /* USER CODE END TIM2_Init 2 */

}

/**
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART1_UART_Init(void)
{

  /* USER CODE BEGIN USART1_Init 0 */

  /* USER CODE END USART1_Init 0 */

  /* USER CODE BEGIN USART1_Init 1 */

  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 9600;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
```

```
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */

  /* USER CODE END USART1_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET);
```

```
  /*Configure GPIO pin : PB3 */
  GPIO_InitStruct.Pin = GPIO_PIN_3;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim2)
{
    if (htim2->Channel == HAL_TIM_ACTIVE_CHANNEL_1)
    {
        if (Is_First_Captured == 0)
        {
            IC_Value1 = HAL_TIM_ReadCapturedValue(htim2, TIM_CHANNEL_1);
            Is_First_Captured = 1;
        }
        else if (Is_First_Captured)
        {
         IC_Value2 = HAL_TIM_ReadCapturedValue(htim2, TIM_CHANNEL_1);
            if (IC_Value2 > IC_Value1)
            {
                Difference = IC_Value2 - IC_Value1;
                Frequency = HAL_RCC_GetPCLK1Freq() / Difference;
                Period_us = 1000000 / Frequency;
                Calcul_Fin = 1;
            }
            else
            {
                Calcul_Fin = 0;
            }
            Is_First_Captured = 0;
        }
    }
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    if (huart == &huart1)
    {
        if (strncmp((char*)rx_buffer, "hauteur", 7) == 0)
        {
            envoie = 1;
        }
        if (strncmp((char*)rx_buffer, "calibre", 7) == 0)
        {
            calibre = 1;
        }
    }
}
/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
```

```
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```