Université de Toulon

IUT de Toulon

Département Génie Electrique et Informatique Industrielle (GEII)

SAE

Tests Unitaires et Integrations

écrit le 18 decembre 2023

par

Bruno HANNA

Encadrant universitaire : Stephane PIGNOL

Table des matières

Introduction

Ce rapport est dédiée à la présentation et à l'analyse des programmes de tests unitaires et tests d'intégration du projet. Nous examinerons les composants suivants :

- **Génération PWM** : Vérification de la capacité du système à générer des signaux PWM précis.
- **Page Web avec Boutons/Slider** : Test de l'interface utilisateur web et de son interaction avec le système.
- **Test d'Intégration** : Évaluation de l'intégration et de la communication entre la page web et le système de génération PWM.
- **Page Web et Génération PWM** : Confirmation de la synchronisation et de la réactivité entre l'interface web et la génération PWM.

En bonus, nous explorerons la liaison TCP, et fournirons une introduction à Node-RED, un outil pour faciliter la création de flux de données entre les différents éléments de notre système.

Tests Unitaires

Dans ce chapitre, nous décrivons les tests unitaires réalisés pour valider chaque composant individuel du système. Les tests sont essentiels pour s'assurer que chaque partie fonctionne correctement avant de procéder à l'intégration des composants.

1.1 Génération PWM

La génération de la modulation de largeur d'impulsion (PWM) est cruciale pour le contrôle précis des périphériques. Dans cette section, nous discutons du code utilisé pour contrôler la PWM via un microcontrôleur ESP8266.

1.1.1 Déclaration des broches

Le code commence par la définition des broches utilisées pour la PWM et pour le relai.

```
1 const int pwmPin = 12;
2 const int relai = 13;
```

Listing 1.1 – Déclaration des broches

La broche numéro 12 est utilisée pour la sortie du signal PWM, et la broche numéro 13 est utilisée pour contrôler le relai.

1.1.2 Fonction de mise à jour de la PWM

La fonction `updatePWM` prend une valeur en entrée et met à jour la sortie PWM en conséquence.

```
1 void updatePWM(int value) {
2   if (value >= 0 && value <= 100) {
3     int pwmValue = map(value, 0, 100, 0, 1023);
4     analogWrite(pwmPin, pwmValue);
5     Serial.println("Valeur PWM mise a jour: " + String(value) + " ( " +
    String(pwmValue) + " )");
6   }
7 }
```

Listing 1.2 – Fonction pour mettre à jour la valeur PWM

Cette fonction mappe la valeur entrée (supposée être entre 0 et 100) à une valeur de PWM entre 0 et 1023, puis met à jour le signal PWM sur la broche correspondante. Elle imprime également la valeur mise à jour sur le moniteur série pour le débogage.

1.1.3 Configuration initiale

La fonction `setup` est utilisée pour initialiser les broches et démarrer la communication série.

```
1 void setup(void) {
2   pinMode(pwmPin, OUTPUT);   // Configurer la broche PWM en sortie
3   pinMode(relai, OUTPUT);    // Configurer la broche RELAI en sortie
4   analogWrite(pwmPin, 0);    // Initialiser la valeur PWM a 0
5   analogWriteResolution(10);// On met la resolution du PWM sur 10 bits
6   analogWriteFreq(30000);    // On met la frequence a 30kHz
7   Serial.begin(115200);      // Demarrer la communication serie
8 }
```

Listing 1.3 – Configuration initiale du microcontrôleur

La résolution du PWM est définie sur 10 bits et la fréquence du signal PWM est réglée sur 30kHz pour obtenir une sortie précise.

1.1.4 Boucle principale

La fonction `loop` vérifie continuellement les données entrantes sur le port série et met à jour la valeur PWM en conséquence.

```
1 void loop(void) {
2   if (Serial.available() > 0) {
3     int value = Serial.parseInt();
4     updatePWM(value);
5   }
6 }
```

Listing 1.4 – Boucle principale pour la lecture du moniteur série et la mise à jour de la PWM

La valeur lue depuis le moniteur série est utilisée pour mettre à jour le signal PWM via la fonction `updatePWM`.

1.1.5 Test de Vérification

L'objectif principal de ce test est de valider la précision et la conformité de la signalisation PWM générée par notre système. Pour cela, nous utilisons un oscilloscope pour mesurer et visualiser les caractéristiques du signal PWM, en particulier sa fréquence et son rapport cyclique. Les spécifications que nous visons dans ce test sont une fréquence de 30 kHz (demander par Mr Leredde) et un rapport cyclique de 28% (choisis aléatoirement).

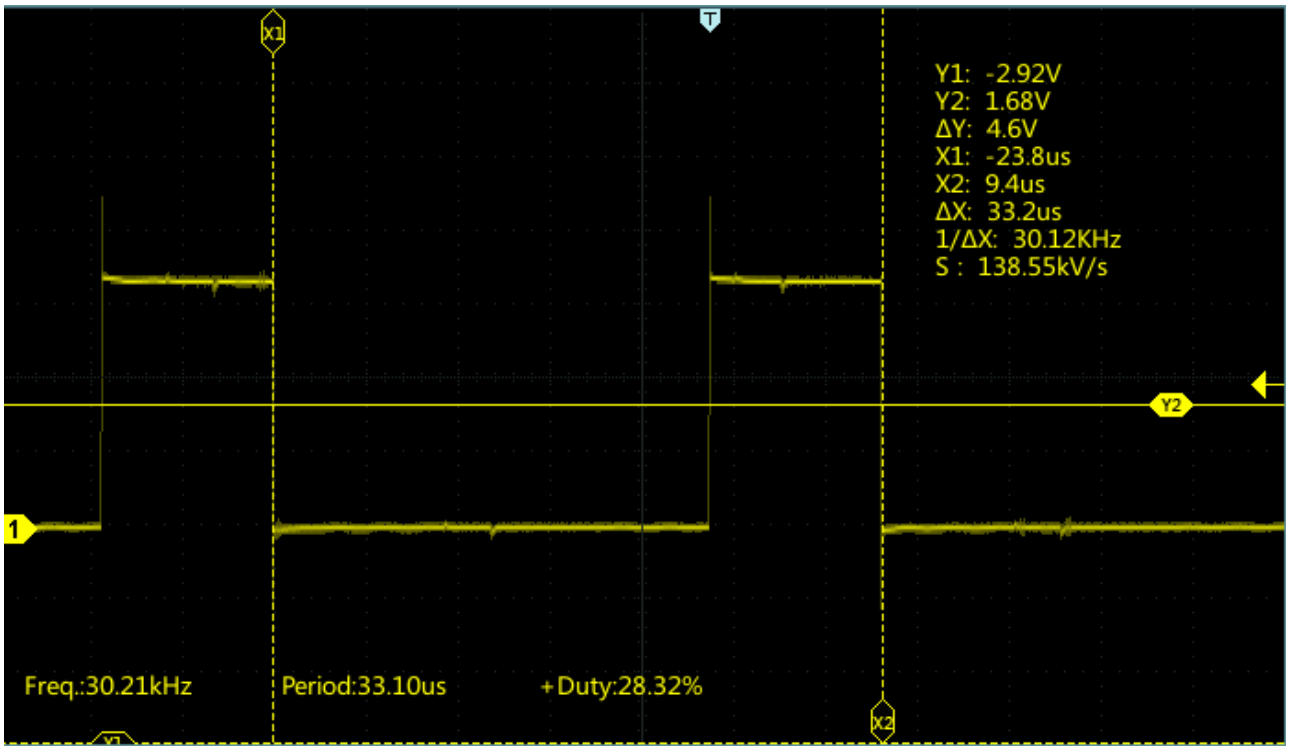


Figure 1.1 – Rendu PWM sur Oscilloscope

1.2 Configuration et Gestion du Serveur Web

Ce programme Arduino configure un serveur web sur un microcontrôleur ESP8266. Le serveur écoute sur le port 80 pour les connexions entrantes, et sert une page web simple.

1.2.1 Inclusions et Définitions

```
#include <ESP8266WiFi.h>  // Inclure la bibliotheque pour gerer le WiFi
#include <WiFiClient.h>   // Inclure la bibliotheque pour gerer les clients WiFi
#include <ESP8266WebServer.h>  // Inclure la bibliotheque pour le serveur Web
#include <ESP8266mDNS.h>  // Inclure la bibliotheque pour le MDNS

// Definition des identifiants WiFi
#ifndef STASSID
#define STASSID "iPhone"
#define STAPSK "framboise"
#endif

const char* ssid = STASSID;       // SSID du reseau WiFi
const char* password = STAPSK;    // Mot de passe du reseau WiFi

```

```
15 ESP8266WebServer server(80);  // Creer une instance du serveur Web sur le
       port 80
```

Listing 1.5 – Inclusions et Definitions

Les bibliotheques pour le WiFi, les clients WiFi, le serveur Web et MDNS sont incluses. Les identifiants du reseau WiFi sont definis. Un objet 'ESP8266WebServer' est cree pour ecouter les connexions entrantes sur le port 80.

1.2.2 Gestion de la Page Racine (Fonction handleRoot)

```
1  void handleRoot() {
2    // HTML pour la page Web
3    String html = "\
4  <html>\
5    <head>\
6      <title>SAE PIGNOL</title>\
7    </head>\
8    <body>\
9      <h1>Test</h1>\
10     <h2>Bruno Hanna et Chloe Muliva</h2>\
11   </body>\
12 </html>";
13   server.send(200, "text/html", html);  // Envoyer la page HTML au client
14 }
```

Listing 1.6 – Gestion de la Page Racine

La fonction 'handleRoot' definit la page racine du serveur web. Lorsqu'un client accede a cette page, un document HTML simple est envoye en reponse. Cette page contient juste un titre et un sous-titre.

1.2.3 Configuration Initiale (Fonction setup)

```
1  void setup(void) {
2    Serial.begin(115200);            // Demarrer la communication serie
3    WiFi.mode(WIFI_STA);             // Configurer le mode WiFi en station
4    WiFi.begin(ssid, password);      // Se connecter au reseau WiFi
5
6    // Attendre la connexion WiFi
7    while (WiFi.status() != WL_CONNECTED) {
8      delay(500);
9      Serial.print(".");
```

```
  }

  // Afficher l'adresse IP une fois connecte
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  // Configurer les routes pour le serveur Web
  server.on("/", handleRoot);
  server.onNotFound([]() {
    server.send(404, "text/plain", "Not found");
  });

  // Demarrer le serveur Web
  server.begin();
  Serial.println("Serveur Web demarre");
}
```

Listing 1.7 – Configuration Initiale

La communication serie est demarree pour le debogage. Le microcontroleur se connecte au reseau WiFi. Une fois connecte, l'adresse IP est affichee. Les routes pour le serveur web sont configurees, y compris la route pour la page racine et une route pour gerer les requetes non trouvees. Enfin, le serveur web est demarre.

1.2.4 Boucle Principale (Fonction loop)

```
void loop(void) {
  server.handleClient();  // Appeler la fonction pour gerer les clients
   Web
}
```

Listing 1.8 – Boucle Principale

La fonction 'loop' appelle en continu 'server.handleClient()', ce qui permet au serveur web de traiter les requetes entrantes, de servir la page racine et de repondre aux requetes non trouvees.

Cette structure detaillee explique comment le serveur web fonctionne sur le microcontroleur ESP8266, comment il gere les requetes des clients, et comment il affiche les informations pour le debogage et le suivi.

1.2.5 Test de Vérification

L'objectif principal de ce test est de valider la précision et la conformité du site web généré par l'ESP8266. Pour cela, il faut se connecter au DNS ou l'adresse IP de la carte ESP (dans notre cas, ouecgreg.local)

Figure 1.2 – Rendu Site Web

1.3 Test Unitaire pour le Serveur TCP et Interaction avec Node-RED

Cette section detaille le fonctionnement du serveur TCP mis en place sur le microcontroleur ESP8266 et decrit l'interaction avec ce serveur via une interface utilisateur creee dans Node-RED.

1.3.1 Inclusions et Definitions

```
1 #include <ESP8266WiFi.h>  // Inclure la bibliotheque pour gerer le WiFi
2 #include <WiFiClient.h>   // Inclure la bibliotheque pour gerer les
     clients WiFi
3
4 // Definition des identifiants WiFi
5 #ifndef STASSID
```

```
#define STASSID "iPhone"
#define STAPSK "framboise"
#endif

const char* ssid = STASSID;        // SSID du reseau WiFi
const char* password = STAPSK;     // Mot de passe du reseau WiFi

WiFiServer tcpServer(81);  // Creer une instance du serveur TCP sur le
    port 81
```

Listing 1.9 – Inclusions et Definitions

Les bibliotheques 'ESP8266WiFi' et 'WiFiClient' sont incluses pour gerer la connectivite WiFi et les clients WiFi, respectivement. Les identifiants du reseau WiFi (SSID et mot de passe) sont definis. Un objet 'WiFiServer' nomme 'tcpServer' est cree pour ecouter sur le port 81.

1.3.2 Configuration Initiale (Fonction setup)

```
void setup(void) {
  Serial.begin(115200);             // Demarrer la communication serie
  WiFi.mode(WIFI_STA);              // Configurer le mode WiFi en station
  WiFi.begin(ssid, password);       // Se connecter au reseau WiFi

  // Attendre la connexion WiFi
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  // Afficher l'adresse IP une fois connecte
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  tcpServer.begin();                // Demarrer le serveur TCP
  Serial.println("Serveur TCP demarre");
}
```

Listing 1.10 – Configuration Initiale

La communication serie est demarree pour le debogage et l'affichage des messages. Le microcontroleur est configure en mode station WiFi et se connecte au reseau specifie. Une boucle attend que

la connexion WiFi soit etablie, affichant des points sur le moniteur serie pour indiquer l'avancement. Une fois connecte, l'adresse IP du microcontroleur sur le reseau est affichee. Le serveur TCP est demarre, ecoutant pour les connexions entrantes sur le port specifie.

1.3.3 Gestion des Clients TCP (Fonction handleTCPClient)

```
1  void handleTCPClient() {
2    static WiFiClient activeTcpClient;  // Client TCP pour maintenir une
       connexion active
3
4    // Verifier si un nouveau client est disponible et si aucun client
       actif n'est deja connecte
5    WiFiClient newClient = tcpServer.available();
6    if (newClient) {
7      if (!activeTcpClient || !activeTcpClient.connected()) {
8        activeTcpClient = newClient;  // Accepter le nouveau client
9        Serial.println("Nouveau client TCP connecte");
10     }
11   }
12
13   // Traiter les donnees si le client est connecte
14   if (activeTcpClient.connected()) {
15     while (activeTcpClient.available()) {
16       String line = activeTcpClient.readStringUntil('\r');  // Lire la
       ligne recue
17       Serial.print("Donnees TCP recues: ");
18       Serial.println(line);
19     }
20   }
21 }
```

Listing 1.11 – Gestion des Clients TCP

Un objet 'WiFiClient' nomme 'activeTcpClient' est utilise pour maintenir la connexion avec le client TCP. La fonction verifie si de nouveaux clients sont disponibles. Si un nouveau client se connecte, il est accepte et un message est affiche. Si des donnees sont disponibles a partir du client connecte, elles sont lues et affichees sur le moniteur serie.

1.3.4 Boucle Principale (Fonction loop)

```
1  void loop(void) {
2    handleTCPClient();  // Appeler la fonction pour gerer les clients TCP
```

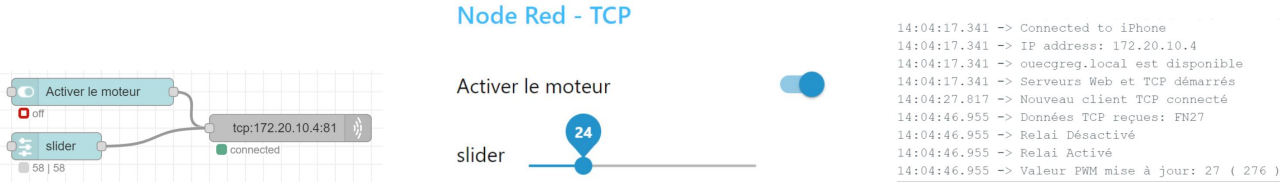
```
3 }
```

Listing 1.12 – Boucle Principale

La fonction ‘loop‘ appelle en continu ‘handleTCPClient‘, ce qui permet au serveur TCP de rester reactif aux nouvelles connexions et aux donnees envoyees par les clients.

1.3.5 Test de Vérification

Les trois photos présentées illustrent les étapes successives de la communication TCP au sein de notre système. Elles démontrent clairement le processus de transmission des données, la réception correcte des signaux par le serveur TCP



(a) Programme Node-Red (b) Interface Node-Red (c) Port série Arduino

Figure 1.3 – Vérification de la réception de donnée TCP

L’objectif principal de ce test est de valider la précision et la conformité de la transmission TCP sur l’ESP8266. Pour cela, il faudrait utiliser wireshark mais n’ayant pas le logiciel je me suis contenter de montrer le bon fonctionnement de la transmission

1.4 Explication Node-RED

Node-RED est un outil de programmation visuelle basé sur un navigateur pour le câblage ensemble des dispositifs matériels, des API, et des services en ligne de manière nouvelle et intéressante. Il est particulièrement bien adapté pour les projets IoT (Internet des Objets) et offre une grande flexibilité et une facilité d’utilisation.

1.4.1 Installation de Node-RED

Node-RED peut être installé sur une variété de systèmes, y compris Windows, MacOS, Linux, et même sur des dispositifs comme le Raspberry Pi. Voici les étapes générales pour l’installation de Node-RED :

1. Assurez-vous que Node.js est installé sur votre système. Node-RED est construit sur Node.js.
2. Installez Node-RED en utilisant npm, le gestionnaire de paquets pour Node.js. Vous pouvez le faire en exécutant la commande suivante dans votre terminal ou invite de commande : `npm install -g -unsafe-perm node-red`.

3. Une fois installé, vous pouvez démarrer Node-RED en exécutant `node-red` dans votre terminal.

1.4.2 Pourquoi Node-RED est une alternative à LabVIEW

Node-RED est souvent comparé à LabVIEW car les deux outils offrent des environnements de développement graphique pour le câblage de systèmes. Cependant, Node-RED offre plusieurs avantages :

— **Gratuité :** Node-RED est un outil open-source et totalement gratuit, tandis que LabVIEW est un produit commercial.

— **Légèreté :** Node-RED est léger et peut être exécuté sur des dispositifs comme le Raspberry Pi, ce qui le rend idéal pour des projets IoT.

— **Communauté :** Node-RED a une communauté active et croissante, offrant un large éventail de nœuds prêts à l'emploi qui peuvent être facilement intégrés dans votre projet.

— **Flexibilité :** Node-RED peut être facilement intégré avec une multitude de services en ligne et de systèmes.

— **Installation de Node-RED Dashboard :** Le flow utilise des nœuds de l'interface utilisateur, tels que 'ui_switch' et 'ui_slider'. Ces nœuds font partie de Node-RED Dashboard, une collection de nœuds qui permettent de créer des interfaces utilisateur dans Node-RED. Pour l'installer, exécutez la commande suivante dans votre terminal ou invite de commande :

```
npm install node-red-dashboard
```

— **Configuration des Nœuds :** Assurez-vous que chaque nœud est correctement configuré. Par exemple :
  — Le nœud 'tcp out' doit avoir l'adresse IP correcte ('host') et le port ('port') du serveur TCP cible.
  — Les nœuds 'ui_switch' et 'ui_slider' doivent être liés à un groupe et un onglet appropriés (par exemple, 'b1a296188fcbf5b2' et '6bb55c7cbccf2fed' respectivement) dans Node-RED Dashboard.

— **Déploiement du Flow :** Une fois que vous avez importé le flow JSON dans Node-RED et que vous avez configuré tous les nœuds, cliquez sur le bouton 'Deploy' pour appliquer vos changements et démarrer le flow.

— **Accès à l'Interface Utilisateur :** Après le déploiement, vous pouvez accéder à l'interface utilisateur générée par Node-RED Dashboard en visitant l'URL suivante dans votre navigateur :

```
http://<your_node-red_ip>:1880/ui
```

1.4.3 Importer un Flow dans Node-RED

Un flow dans Node-RED est essentiellement une collection de nœuds configurés pour effectuer certaines tâches. Voici comment vous pouvez importer un flow :

1. Ouvrez l'interface utilisateur de Node-RED dans votre navigateur web.
2. Cliquez sur le menu en haut à droite (les trois lignes horizontales), puis sélectionnez 'Import'.
3. Vous pouvez maintenant coller le JSON de votre flow ou le télécharger depuis un fichier.
4. Après avoir collé ou téléchargé le JSON, cliquez sur 'Import' pour ajouter le flow à votre espace de travail Node-RED.

En suivant ces étapes, vous pouvez facilement ajouter de nouveaux flows à votre environnement Node-RED et commencer à les utiliser immédiatement.

Test d'Intégration

Cette section décrit la mise en place et le fonctionnement d'un serveur Web et d'un serveur TCP sur un microcontrôleur ESP8266, détaillant le rôle de chaque partie du code.

2.1 Configuration du Serveur Web et TCP

2.1.1 Inclusions et Définitions

```
#include <ESP8266WiFi.h>  // Inclure la bibliotheque pour gerer le WiFi
#include <WiFiClient.h>   // Inclure la bibliotheque pour gerer les clients WiFi
#include <ESP8266WebServer.h>  // Inclure la bibliotheque pour le serveur Web
#include <ESP8266mDNS.h>  // Inclure la bibliotheque pour le MDNS

// Definition des identifiants WiFi
#ifndef STASSID
#define STASSID "iPhone"
#define STAPSK "framboise"
#endif

const char* ssid = STASSID;       // SSID du reseau WiFi
const char* password = STAPSK;    // Mot de passe du reseau WiFi

ESP8266WebServer server(80);  // Creer une instance du serveur Web sur le port 80
WiFiServer tcpServer(81);     // Creer une instance du serveur TCP sur le port 81
```

Listing 2.1 – Inclusions et Definitions

Le code commence par inclure les bibliotheques nécessaires pour la connectivité WiFi, la gestion des clients WiFi, le serveur Web et le MDNS. Il définit ensuite les identifiants WiFi et crée des instances pour le serveur Web et le serveur TCP.

2.1.2 Serveur Web : Gestion de la Page Racine

```
void handleRoot() {
```

```
2   int lastValue = 0;
3   if (server.hasArg("pwmValue")) {
4     lastValue = server.arg("pwmValue").toInt();  // Recuperer la derniere
      valeur PWM du formulaire
5   }
6
7   // HTML pour la page Web
8   String html = "\
9 <html>\
10  <head>\
11    <title>SAE PIGNOL</title>\
12  </head>\
13  <body>\
14    <h1>PWM Control</h1>\
15    <form action=\"/setpwm\" method=\"POST\">\
16      <input type=\"range\" name=\"pwmValue\" min=\"0\" max=\"100\" value
    =\""
17                + String(lastValue) + "\" oninput=\"updateTextInput(this.
    value)\" style=\"width: 20%;\">\
18      <input type=\"text\" style=\"width: 5%;\" id=\"textInput\" readonly
    >\
19      <input type=\"submit\" value=\"Set PWM\">\
20    </form>\
21    <h2>Bruno Hanna</h2>\
22  </body>\
23 </html>";
24  server.send(200, "text/html", html);  // Envoyer la page HTML au client
25 }
```

Listing 2.2 – Serveur Web : Gestion de la Page Racine

La fonction ‘handleRoot‘ sert la page racine du serveur web. Elle vérifie si une valeur PWM est passée en argument et affiche une page HTML simple permettant de contrôler la PWM.

2.1.3 Serveur Web et TCP : Configuration Initiale

```
1 void setup(void) {
2   Serial.begin(115200);             // Demarrer la communication serie
3   WiFi.mode(WIFI_STA);              // Configurer le mode WiFi en station
4   WiFi.begin(ssid, password);       // Se connecter au reseau WiFi
5
6   // Attendre la connexion WiFi
```

```
7   while (WiFi.status() != WL_CONNECTED) {
8     delay(500);
9     Serial.print(".");
10  }
11
12  // Afficher l'adresse IP une fois connecte
13  Serial.println("");
14  Serial.print("Connected to ");
15  Serial.println(ssid);
16  Serial.print("IP address: ");
17  Serial.println(WiFi.localIP());
18
19  // Configurer les routes pour le serveur Web
20  server.on("/", handleRoot);
21  server.on("/setpwm", handleSetPWM);
22
23  server.onNotFound([]() {
24    server.send(404, "text/plain", "Not found");
25  });
26
27  // Demarrer le serveur Web et TCP
28  server.begin();
29  tcpServer.begin();
30  Serial.println("Serveurs Web et TCP demarres");
31 }
```

Listing 2.3 – Serveur Web et TCP : Configuration Initiale

Dans la fonction ‘setup‘, le microcontrôleur se connecte au réseau WiFi et configure le serveur Web pour gérer la page racine et les requêtes non trouvées. Les serveurs Web et TCP sont démarrés.

2.1.4 Serveur Web : Gestion de la Mise à Jour PWM

```
1 void handleSetPWM() {
2   if (server.hasArg("pwmValue")) {
3     int pwmValue = server.arg("pwmValue").toInt();  // Recuperer la
    valeur PWM du formulaire
4     pwmValue = map(pwmValue, 0, 100, 0, 1023);      // Mapper la valeur
    de 0-100 a 0-1023
5     analogWrite(pwmPin, pwmValue);                  // Mettre a jour le
    signal PWM
6   }
```

```
  server.sendHeader("Location", "/");  // Rediriger vers la page racine
  server.send(303);                    // Envoyer une r ponse de
   redirection
}
```

Listing 2.4 – Serveur Web : Gestion de la Mise à Jour PWM

La fonction ‘handleSetPWM‘ gère la mise à jour de la valeur PWM via le serveur Web. Elle récupère la valeur PWM du formulaire, la mappe à la plage requise et met à jour le signal PWM.

2.1.5 Serveur TCP : Gestion des Clients TCP

```
void handleTCPClient() {
  static WiFiClient activeTcpClient;  // Client TCP pour maintenir une
   connexion active

  // Verifier si un nouveau client est disponible et si aucun client
   actif n'est d j  connecte
  WiFiClient newClient = tcpServer.available();
  if (newClient) {
    if (!activeTcpClient || !activeTcpClient.connected()) {
      activeTcpClient = newClient;  // Accepter le nouveau client
      Serial.println("Nouveau client TCP connecte");
    }
  }

  // Traiter les donnees si le client est connecte
  if (activeTcpClient.connected()) {
    while (activeTcpClient.available()) {
      String line = activeTcpClient.readStringUntil('\r');  // Lire la
   ligne recue
      Serial.print("Donnees TCP recues: ");
      Serial.println(line);

      // Analyser la ligne recue pour les commandes et les chiffres
      for (int i = 0; i < line.length(); i++) {
        char ch = line.charAt(i);
        if (ch == 'N') {
          updateRelai(1);  // Commande pour activer le relai
        } else if (ch == 'F') {
          updateRelai(0);  // Commande pour desactiver le relai
        } else if (isdigit(ch)) {
```

```
28            String numStr = "";
29            while (i < line.length() && isdigit(line.charAt(i))) {
30              numStr += line.charAt(i);
31              i++;
32            }
33            int pwmValue = numStr.toInt();  // Convertir la chaine de
    chiffres en entier
34            updatePWM(pwmValue);            // Mettre a jour la PWM
35            break;                          // Sortir de la boucle apres
    avoir traite les chiffres
36          }
37        }
38      }
39    }
40 }
```

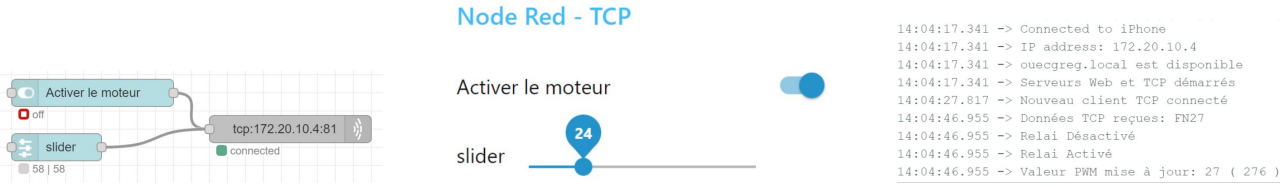
Listing 2.5 – Serveur TCP : Gestion des Clients TCP

La fonction ‘handleTCPClient‘ gère les connexions entrantes des clients TCP. Elle lit les données reçues, traite les commandes pour activer/désactiver le relai ou mettre à jour la valeur PWM.

La structure de ce code offre un cadre pour intégrer un serveur Web et un serveur TCP dans un même système sur un microcontrôleur ESP8266, permettant des interactions via le réseau WiFi pour le contrôle et la surveillance à distance.

2.2 Test de Vérification

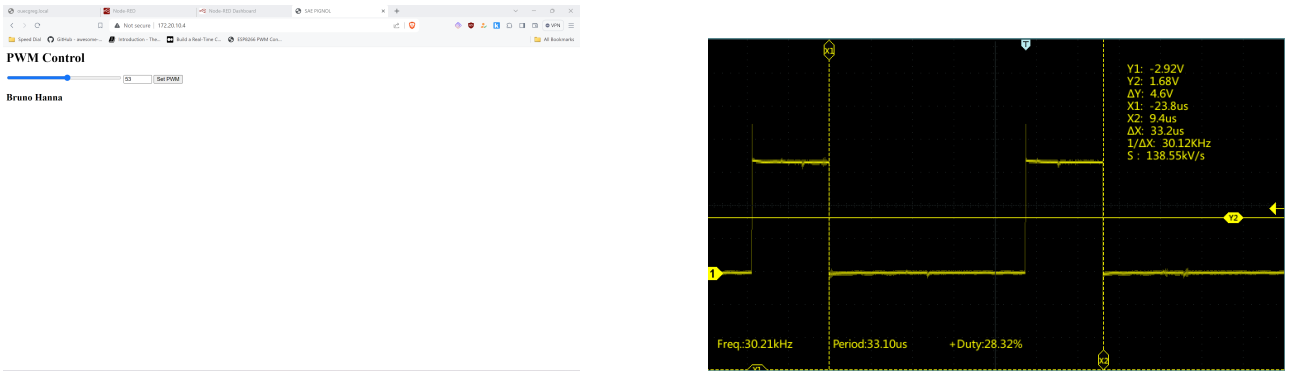
La validation de la fonctionnalité complète du programme nécessite une vérification de la communication TCP. Une série d'images sera utilisé pour illustrer visuellement le succès de ces tests. Les trois premières images ci-dessous montrent clairement que la connexion TCP est bien établie et fonctionne comme prévu :



(a) Programme Node-Red (b) Interface Node-Red (c) Port série Arduino

Figure 2.1 – Vérification de la réception de donnée TCP

Ensuite, nous réitérons le processus, mais cette fois en intégrant le site web pour initier la commande :



(a) Interface WEB (b) Relevé PWM sur Oscilloscope

Figure 2.2 – Vérification du fonctionnement du Site Web (photos misent en annexe)

Ces tests montrent que le site web et le serveur TCP fonctionnent de manière synchrone sur la même carte, confirmant ainsi le respect du cahier des charges.

Code Source Complet

Les annexes suivantes contiennent les listes complètes des codes sources utilisés dans le cadre de ce projet. Ces codes représentent les composantes essentielles du système et illustrent les implémentations spécifiques des fonctionnalités discutées dans les chapitres précédents.

A.1 Code Arduino pour la PWM

```
#include <ESP8266WiFi.h>

const int pwmPin = 12;  // Definir le numero de la broche utilisee pour le signal PWM
const int relai = 13;   // Definir le numero de la broche utilisee pour le relai

// Fonction pour mettre a jour la valeur PWM
void updatePWM(int value) {
  if (value >= 0 && value <= 100) {
    int pwmValue = map(value, 0, 100, 0, 1023);  // Mapper la valeur de 0-100 a 0-1023
    analogWrite(pwmPin, pwmValue);               // Mettre a jour le signal PWM
    Serial.println("Valeur PWM mise a jour: " + String(value) + " ( " + String(pwmValue) + " )");
  }
}

void setup(void) {
  pinMode(pwmPin, OUTPUT);  // Configurer la broche PWM en sortie
  pinMode(relai, OUTPUT);   // Configurer la broche RELAI en sortie
  analogWrite(pwmPin, 0);   // Initialiser la valeur PWM a 0
  analogWriteResolution(10); // On met la resolution du PWM sur 10 bits (1024 valeurs)
  analogWriteFreq(30000);   // On met la frequence a 30kHz

  Serial.begin(115200);     // Demarrer la communication serie
}
```

```
24
25 void loop(void) {
26   if (Serial.available() > 0) {
27     int value = Serial.parseInt();  // Lire la valeur entree dans le
     moniteur serie
28     updatePWM(value);  // Mettre a jour la PWM avec la valeur lue
29   }
30 }
```

Listing A.1 – Code Arduino complet pour la PWM

A.2 Code Arduino pour le Serveur Web

```
1 #include <ESP8266WiFi.h>
2 #include <WiFiClient.h>
3 #include <ESP8266WebServer.h>
4 #include <ESP8266mDNS.h>
5
6 // Definir les identifiants WiFi
7 #ifndef STASSID
8 #define STASSID "iPhone"
9 #define STAPSK "framboise"
10 #endif
11
12 const char* ssid = STASSID;
13 const char* password = STAPSK;
14
15 ESP8266WebServer server(80);  // Creer une instance du serveur Web sur le
     port 80
16
17 // Fonction pour gerer la page racine du serveur Web
18 void handleRoot() {
19   // HTML pour la page Web
20   String html = "\
21 <html>\
22   <head>\
23     <title>SAE PIGNOL</title>\
24   </head>\
25   <body>\
26     <h1>Test</h1>\
27     <h2>Bruno Hanna</h2>\
```

```
  </body>\
</html>";
  server.send(200, "text/html", html);  // Envoyer la page HTML au client
}

void setup(void) {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);  // Se connecter au reseau WiFi

  // Attendre la connexion WiFi
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  // Afficher l'adresse IP une fois connecte
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  // Configurer les routes pour le serveur Web
  server.on("/", handleRoot);
  server.onNotFound([]() {
    server.send(404, "text/plain", "Not found");
  });

  // Demarrer le serveur Web
  server.begin();
  Serial.println("Serveur Web demarre");
}

void loop(void) {
  server.handleClient();  // Gerer les clients Web
}
```

Listing A.2 – Code Arduino complet pour le Serveur Web

A.3 Code Arduino pour le Serveur TCP

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>

// Definition des identifiants WiFi
#ifndef STASSID
#define STASSID "iPhone"
#define STAPSK "framboise"
#endif

const char* ssid = STASSID;
const char* password = STAPSK;

WiFiServer tcpServer(81);      // Creer une instance du serveur TCP sur le
    port 81

void setup(void) {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);  // Se connecter au reseau WiFi

  // Attendre la connexion WiFi
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  // Afficher l'adresse IP une fois connecte
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  // Demarrer le serveur TCP
  tcpServer.begin();
  Serial.println("Serveur TCP demarre");
}

// Fonction pour gerer les clients TCP
```

```
39 void handleTCPClient() {
40   static WiFiClient activeTcpClient;  // Client TCP pour maintenir une
      connexion active
41
42   // Verifier si un nouveau client est disponible et si aucun client
      actif n'est deja connecte
43   WiFiClient newClient = tcpServer.available();
44   if (newClient) {
45     if (!activeTcpClient || !activeTcpClient.connected()) {
46       activeTcpClient = newClient;  // Accepter le nouveau client
47       Serial.println("Nouveau client TCP connecte");
48     }
49   }
50
51   // Traiter les donnees si le client est connecte
52   if (activeTcpClient.connected()) {
53     while (activeTcpClient.available()) {
54       String line = activeTcpClient.readStringUntil('\r');  // Lire la
      ligne recue
55       Serial.print("Donnees TCP recues: ");
56       Serial.println(line);
57     }
58   }
59 }
60
61 void loop(void) {
62   handleTCPClient();      // Gerer les clients TCP
63 }
```

Listing A.3 – Code Arduino pour le Serveur TCP

A.4 Code JSON pour Node-RED

```
1 [
2     {
3         "id": "333c7c7a74219384",
4         "type": "tab",
5         "label": "SAE Pignol",
6         "disabled": false,
7         "info": "",
8         "env": []
```

```
    },
    {
        "id": "4",
        "type": "tcp out",
        "z": "333c7c7a74219384",
        "name": "",
        "host": "172.20.10.14",
        "port": "81",
        "beserver": "client",
        "base64": false,
        "end": false,
        "tls": "",
        "x": 560,
        "y": 220,
        "wires": []
    },
    {
        "id": "81b7de654c27a493",
        "type": "ui_switch",
        "z": "333c7c7a74219384",
        "name": "",
        "label": "Activer le moteur",
        "tooltip": "",
        "group": "b1a296188fcbf5b2",
        "order": 1,
        "width": 0,
        "height": 0,
        "passthru": true,
        "decouple": "false",
        "topic": "switch",
        "topicType": "msg",
        "style": "",
        "onvalue": "N",
        "onvalueType": "str",
        "onicon": "",
        "oncolor": "",
        "offvalue": "F",
        "offvalueType": "str",
        "officon": "",
        "offcolor": "",
        "animate": false,
```

```
        "className": "",
        "x": 330,
        "y": 180,
        "wires": [
            [
                "4"
            ]
        ]
    },
    {
        "id": "9c5fc49dd8f72b8b",
        "type": "ui_slider",
        "z": "333c7c7a74219384",
        "name": "",
        "label": "slider",
        "tooltip": "",
        "group": "b1a296188fcbf5b2",
        "order": 1,
        "width": "5",
        "height": "1",
        "passthru": false,
        "outs": "end",
        "topic": "pwm",
        "topicType": "msg",
        "min": 0,
        "max": "100",
        "step": 1,
        "className": "",
        "x": 290,
        "y": 240,
        "wires": [
            [
                "4"
            ]
        ]
    },
    {
        "id": "b1a296188fcbf5b2",
        "type": "ui_group",
        "name": "Node Red - TCP",
        "tab": "6bb55c7cbccf2fed",
```

```
        "order": 1,
        "disp": true,
        "width": "6",
        "collapse": false,
        "className": ""
    },
    {
        "id": "6bb55c7cbccf2fed",
        "type": "ui_tab",
        "name": "SAE Pignol",
        "icon": "dashboard",
        "disabled": false,
        "hidden": false
    }
]
```

Listing A.4 – Configuration Node-RED a importer

A.5 Code Arduino pour le Serveur Web et TCP

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>

// Definir les identifiants WiFi
#ifndef STASSID
#define STASSID "iPhone"
#define STAPSK "framboise"
#endif

const char* ssid = STASSID;
const char* password = STAPSK;

ESP8266WebServer server(80);  // Creer une instance du serveur Web sur le port 80
WiFiServer tcpServer(81);     // Creer une instance du serveur TCP sur le port 81

const int pwmPin = 12;  // Definir le numero de la broche utilisee pour le signal PWM
```

```
19 const int relai = 13;   // Definir le numero de la broche utilisee pour
     le relai
20
21 void handleRoot() {
22   int lastValue = 0;
23   if (server.hasArg("pwmValue")) {
24     lastValue = server.arg("pwmValue").toInt();  // Recuperer la derniere
     valeur PWM du formulaire
25   }
26
27   String html = "\
28 <html>\
29   <head>\
30     <title>SAE PIGNOL</title>\
31     <script>\
32       function updateTextInput(val) {\
33         document.getElementById('textInput').value=val;\
34       }\
35     </script>\
36   </head>\
37   <body>\
38     <h1>PWM Control</h1>\
39     <form action=\"/setpwm\" method=\"POST\">\
40       <input type=\"range\" name=\"pwmValue\" min=\"0\" max=\"100\" value
     =\""
41                + String(lastValue) + "\" oninput=\"updateTextInput(this.
     value)\" style=\"width: 20%;\">\
42       <input type=\"text\" style=\"width: 5%;\" id=\"textInput\" readonly
     >\
43       <input type=\"submit\" value=\"Set PWM\">\
44     </form>\
45     <h2>Bruno Hanna et Chloe Muliva</h2>\
46   </body>\
47 </html>";
48   server.send(200, "text/html", html);  // Envoyer la page HTML au client
49 }
50
51 void handleSetPWM() {
52   if (server.hasArg("pwmValue")) {
53     int pwmValue = server.arg("pwmValue").toInt();  // Recuperer la
     valeur PWM du formulaire
```

```
54      pwmValue = map(pwmValue, 0, 100, 0, 1023);      // Mapper la valeur
    de 0-100 a 0-1023
55      analogWrite(pwmPin, pwmValue);                  // Mettre a jour le
    signal PWM
56    }
57    server.sendHeader("Location", "/");  // Rediriger vers la page racine
58    server.send(303);                    // Envoyer une reponse de
    redirection
59  }
60
61  void updatePWM(int value) {
62    if (value >= 0 && value <= 100) {
63      int pwmValue = map(value, 0, 100, 0, 1023);  // Mapper la valeur de
    0-100 a 0-1023
64      analogWrite(pwmPin, pwmValue);               // Mettre a jour le
    signal PWM
65      Serial.println("Valeur PWM mise a jour: " + String(value) + " ( " +
    String(pwmValue) + " )");
66    }
67  }
68
69  void updateRelai(bool value) {
70    if (value) {
71      digitalWrite(relai, value);  // Mettre a jour le relai
72      Serial.println("Relai Active");
73    } else if (!(value)) {
74      digitalWrite(relai, value);  // Mettre a jour le relai
75      Serial.println("Relai Desactive");
76    }
77  }
78
79  void handleTCPClient() {
80    static WiFiClient activeTcpClient;  // Client TCP pour maintenir une
    connexion active
81
82    WiFiClient newClient = tcpServer.available();
83    if (newClient) {
84      if (!activeTcpClient || !activeTcpClient.connected()) {
85        activeTcpClient = newClient;  // Accepter le nouveau client
86        Serial.println("Nouveau client TCP connecte");
87      }
```

```
88    }
89
90    if (activeTcpClient.connected()) {
91      while (activeTcpClient.available()) {
92        String line = activeTcpClient.readStringUntil('\r');  // Lire la
      ligne recue
93        Serial.print("Donnees TCP recues: ");
94        Serial.println(line);
95
96        for (int i = 0; i < line.length(); i++) {
97          char ch = line.charAt(i);
98          if (ch == 'N') {
99            updateRelai(1);  // Commande pour activer le relai
100         } else if (ch == 'F') {
101           updateRelai(0);  // Commande pour desactiver le relai
102         } else if (isdigit(ch)) {
103           String numStr = "";
104           while (i < line.length() && isdigit(line.charAt(i))) {
105             numStr += line.charAt(i);
106             i++;
107           }
108           int pwmValue = numStr.toInt();  // Convertir la chaine de
      chiffres en entier
109           updatePWM(pwmValue);            // Mettre a jour la PWM
110           break;                          // Sortir de la boucle apres
      avoir traite les chiffres
111         }
112       }
113     }
114   }
115 }
116
117 void setup(void) {
118   pinMode(pwmPin, OUTPUT);   // Configurer la broche PWM en sortie
119   pinMode(relai, OUTPUT);    // Configurer la broche RELAI en sortie
120   analogWrite(pwmPin, 0);    // Initialiser la valeur PWM a 0
121   analogWriteResolution(10);// On met la resolution du PWM sur 10 bits
      (1024 valeurs)
122   analogWriteFreq(30000);    // On met la frequence a 30kHz
123
124   Serial.begin(115200);
```

```
125   WiFi.mode(WIFI_STA);
126   WiFi.begin(ssid, password);  // Se connecter au reseau WiFi
127
128   // Attendre la connexion WiFi
129   while (WiFi.status() != WL_CONNECTED) {
130     delay(500);
131     Serial.print(".");
132   }
133
134   // Afficher l'adresse IP une fois connecte
135   Serial.println("");
136   Serial.print("Connected to ");
137   Serial.println(ssid);
138   Serial.print("IP address: ");
139   Serial.println(WiFi.localIP());
140
141   // Demarrer le serveur MDNS
142   if (MDNS.begin("ouecgreg")) { Serial.println("ouecgreg.local est
      disponible"); }
143
144   // Configurer les routes pour le serveur Web
145   server.on("/", handleRoot);
146   server.on("/setpwm", handleSetPWM);
147   server.onNotFound([]() {
148     server.send(404, "text/plain", "Not found");
149   });
150
151   // Demarrer les serveurs Web et TCP
152   server.begin();
153   tcpServer.begin();
154
155   Serial.println("Serveurs Web et TCP demarres");
156 }
157
158 void loop(void) {
159   server.handleClient();  // Gerer les clients Web
160   MDNS.update();          // Mettre a jour le service MDNS
161   handleTCPClient();      // Gerer les clients TCP
162 }
```

Listing A.5 – Code Arduino pour le Serveur Web et TCP

A.6 Image

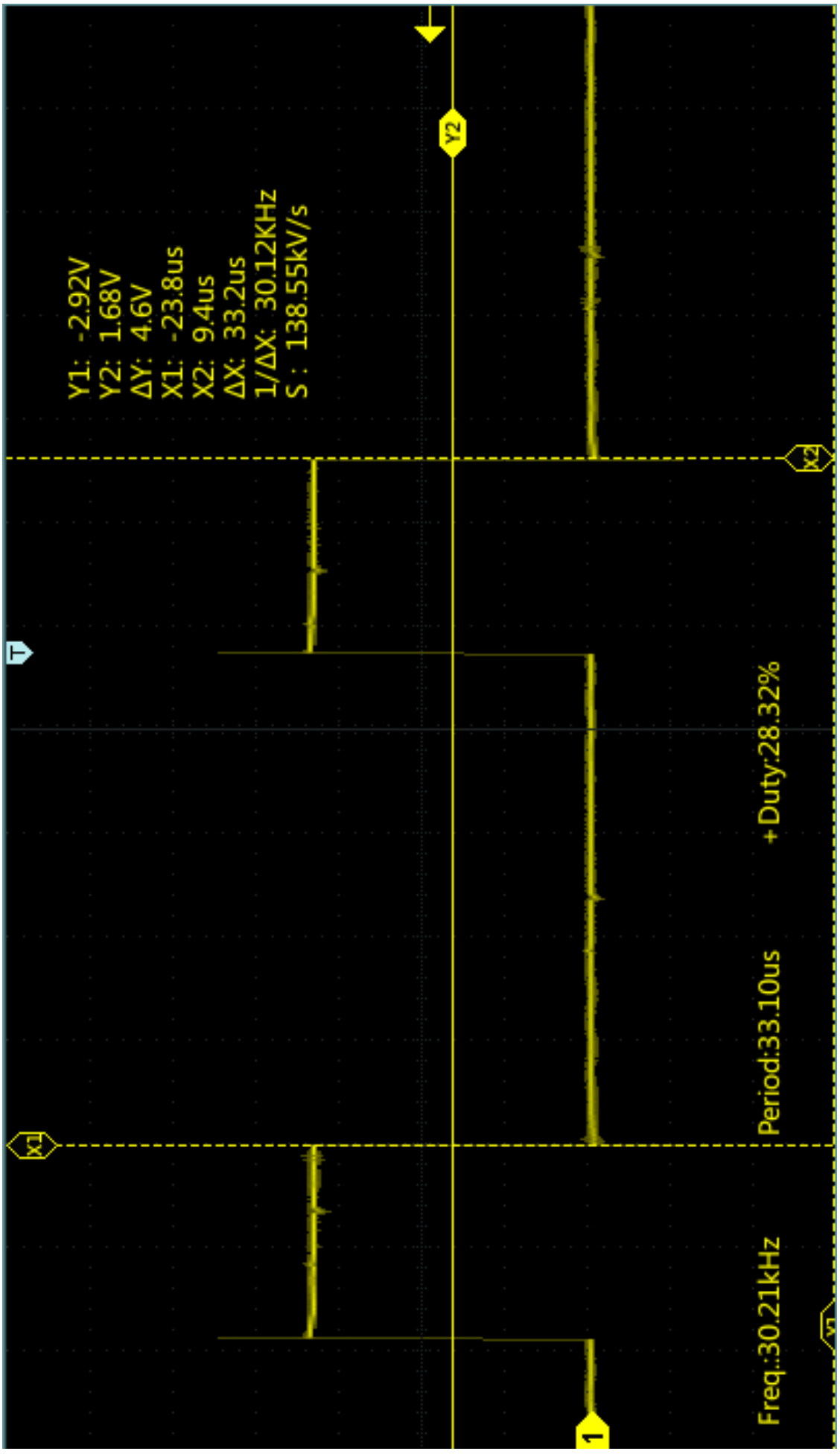


Figure A.1 – Relevé PWM sur Oscilloscope

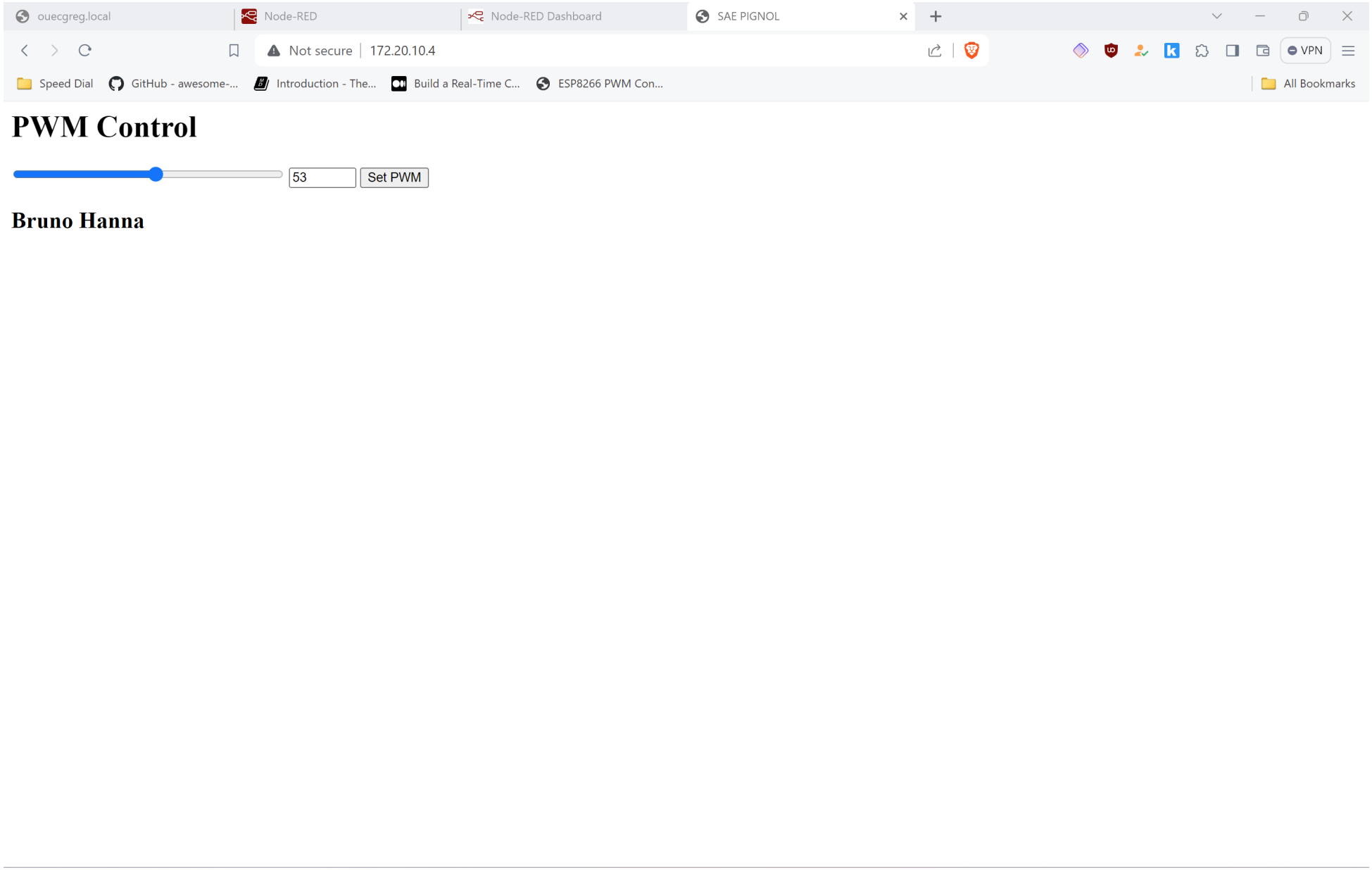


Figure A.2 – Site WEB