TUTORIEL : VOITURE CONTRÔLABLE PAR TELEPHONE

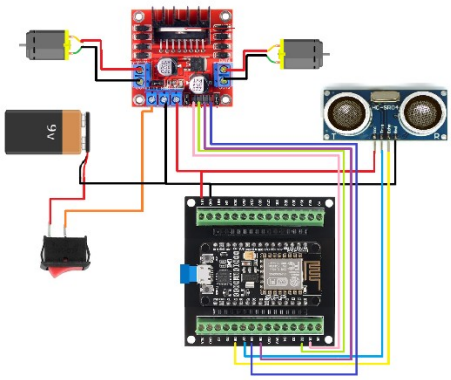
1) Matériel nécessaire :

Pour ce montage vous aurez besoin :

- D'une carte Arduino (ici esp8266 avec adaptateur)
- D'une alimentation externe suffisamment puissance pour vos moteurs (de 12V maximum pour notre matériel, celle utilisé lors de notre projet était une pile 9V classique)
- D'un contrôle moteur L298N (un autre contrôle moteur peut être nécessaire si alimentation à plus de 12V)
- De 2 moteurs avec des roues adaptées.
- D'un bouton on/off.
- D'un capteur de distance à ultrasons.
- De fils de connexions mâle-mâle et mâle-femelle.

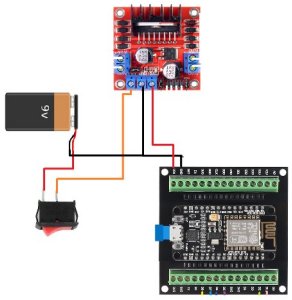
2) Montage à faire :

Le montage est le suivant :

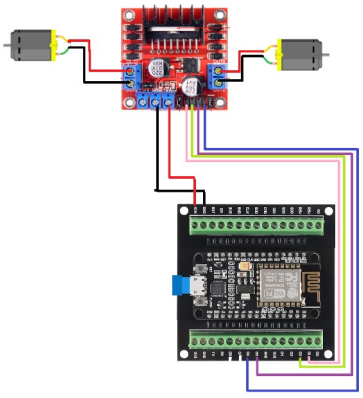


Il est constitué de 3 parties principales que je vais détaillées :

Tout d'abord l'alimentation qui est contrôlée par un bouton, et qui part dans les prises 12V et GND du contrôle moteur avant de ressortir par les prise 5V et GND du contrôle moteur pour aller alimenter la carte par les prise VIN (5V) et G (GND)

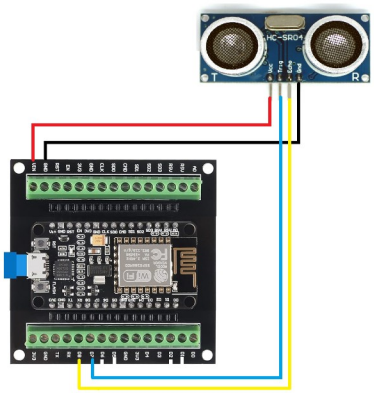


On a ensuite les moteurs reliés de chaque côté du contrôle moteur. En faisant bien attention d'avoir les fils connéctés dans le même sens de chaque coté (par exemple ici fil rouge sur OUT1 et OUT3 et fils noirs sur OUT 2 et 4) .
Ces moteurs sont contrôlés par les entrées de la carte qui sont les 4 fils connectés au IN1,2,3 et 4 (1 à 4 de gauche à droite si pas afffichés) chacun contrôlant le « OUT » du même numéros.
Et permettant donc de contrôler le sens de rotation des roues en contrôlant le sens du courant dans les moteurs.



Enfin, on a la partie capteur ultrason, qui sera diriger vers le sol et permettra d'empêcher à la voiture de rouler dans un vide devant elle, en detectant tout changement soudain de distance au sol.

Il est important de noté que ce dernier doit se trouver à une distance suffisante de vos roues afin que ces dernières est le temps de s'arreter avant le vide.



3) Explication du code :

On commence comme d'habitude avec les codes Arduino avec la parties des bibliothèques nécessaire. Ici il y en a 3:

Tout d'abord ESP8266WiFi.h :

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

Qui permet de contrôler la connexion à, ou l'émission d'une connexion Wi-Fi avec une carte Arduino. (ici une esp8266, chaque controlleur arduino ayant une carte Wi-Fi aura sa propre bibliothèques dédiées)

Ensuite la bibliothèque ESPASynchWebServer.h :

```
#include <ESPAsyncWebServer.h>
```

Qui permet de contrôler un serveur web depuis la carte Arduino, et donc dans notre cas d'héberger le site internet qui nous permettra de contrôler la voiture.

Puis la bibliothèque DNSServer.h :

```
#include <DNSServer.h>
```

Qui permet de changer l'IP du site internet qui peut parfois ne pas être simple à se rappeler à long termes, par une adresse standard du type « voiture.com », ou bien rediriger toute les adresses vers notre site une fois connecter à la voiture.

On va ensuite définir toute les prises utiles pour contrôler notre voiture. On en aura ici besoin de 4 car on a 2 moteur chacun possédant 2 sens de rotation :

```
#define IN1 D1 // GPIO5
#define IN2 D2 // GPIO4
#define IN3 D5 // GPIO14
#define IN4 D6 // GPIO12
```

On peut savoir quelle prise correspond à quel sens de rotations en faisant correspondre nos « IN » (informations que l'on envoie) aux « OUT » (sortie qui enverront un courant dans les moteurs) sur le contrôle moteur. Généralement 1/2 pour la roue de droite et 3/4 pour celle de gauche.
On peut alors déduire le sens de passage du courant dans le moteur et donc son sens de rotation du moteur correspondant à chaque prise « IN ».

On défini ensuite les prise de notre capteur ultrason :

```
#define TRIGGER_PIN D7 // GPIO13

#define ECHO_PIN D8    // GPIO15
```

Le premier (TRIGGER) permet de définir le moment de l'envoie d'une mesure et le 2nd (ECHO) permet de calculer le temps que le signal ultrason met à revenir, et donc d'en déduire la distance entre le capteur et la surface la plus proche car la vitesse des ultrasons est connue.

On défini ensuite 3 variables qui serviront au fonctionnement du capteur.
Tout d'abord la distance maximale que l'on va tolérer avec l'objet (ici le sol)

```
const float MAX_DISTANCE = 5.0;
```

Dans un usage standard on définira souvent une distance minimale en dessous de laquelle on ou voudras s'éloigner d'une obstacle. Mais ici notre capteur est pointé en direction du sol et sert à arrêter la voiture si elle approche d'un vide important , d'où l'utilisation d'une distance maximale par rapport au sol.

On ajoute à cela 2 variables qui serviront à définir le temps depuis la dernière mesure (0 par défaut)

```
unsigned long lastMeasurement = 0;
```

Et l'écart souhaité (en milliseconde) entre les mesures :

```
const unsigned long MEASURE_INTERVAL = 10;
```

On défini ensuite le nom et mot de passe du Wi-Fi qui sera généré par la carte afin de nous connecter au site de la voiture.

```
const char* ssid = "ESP-Robot";

const char* password = "";
```

Et enfin créer l'adresse IP qui serais utiliser par défaut pour le site internet si il n'y avais pas de DNS :

```
IPAddress apIP(192, 168, 4, 1);
```

On déclare et donne son nom à notre serveur DNS

```
DNSServer dnsServer;
```

Et on fait de même pour notre serveur web qui permettra d'héberger le site internet (80 correspond au port standard de communication HTTP)

```
AsyncWebServer server(80);
```

On rentre maintenant dans le void setup , qui sera compiler une seule fois lors du lancement de la carte :

```
void setup() {
```

On définis d'abords le type de chaque prise défini plus haut , tout d'abord les sorties de la carte sur laquelle on va envoyer des informations:

```
pinMode(IN1, OUTPUT);
pinMode(IN2, OUTPUT);
pinMode(IN3, OUTPUT);
pinMode(IN4, OUTPUT);
pinMode(TRIGGER_PIN, OUTPUT);
```

et les entrées sur lesquelles la carte va recevoir des informations :

```
pinMode(ECHO_PIN, INPUT);
```

On met ensuite en place la connexion Wi-Fi tout d'abord en donnant le noms et mot de passe qu'on a choisi.

```
WiFi.softAP(ssid, password);
```

Ensuite on met en place cette connexion en donnant son IP locale, qui servira d'adresse IP à la carte. Puis l'IP « Gateway » qui permet le trafic d'informations dans le réseau (ici aucune raison de les différenciées, cependant ça peut être le cas dans une réseau à plusieurs composant) et enfin le masque de sous-réseaux qui va permettre de définir le nombre d'adresse IP disponible dans le réseau (Ici les 3 premiers octets sont fixe et seul le 4ème est dispo soit 256 IP possibles)

```
WiFi.softAPConfig(apIP, apIP, IPAddress(255, 255, 255, 0));
```

Et enfin on lance notre DNS avec la fonction dnsServer.start.
Le nombre 53 correspond au port standard pour que le DNS fonctionne. Entre les '' '' , on trouve le nom de site que l'on souhaite (mettre ''*'' permet de faire en sorte que tout les adresses renvoie à la même adresse IP). Et enfin on donne l'adresse IP du site que l'on souhaite avoir pour le contrôle de la voiture.

```
dnsServer.start(53, "*", apIP);
```

On arrête ensuite le les moteurs de la voiture afin de s'assurer de bien contrôler la voiture lors qu'on la lance.

```
stopMotors();
```

Cette fonction , comme tout les autres non définie dans les bibliothèques, est définie à la main en dans le code :

```
void stopMotors() {
```

Comme toute les autres fonctions qui définissent des actions sur les moteurs, c'est une simple suite d'information à envoyer les pin IN1 à IN4 du contrôle moteur.
Ici tout est mis à la valleur «LOW» (pas de courant envoyer) afin de stopper toute rotations des moteurs.

```
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
}
```

On définit ensuite la marche à suivre pour le serveur sur chaque action qu'il peut recevoir.

Tout d'abord sur une information vide (''/'') :

```
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) {
```

Alors le serveur doit prendre la fonction getHTMLPage(), afin qu'il puisse afficher le site internet que l'on souhaite.

```
    request->send(200, "text/html", getHTMLPage());
  });
```

Cette fonction retourne une chaîne de caractère contenant le code HTML correspondant au site internet que l'on souhaite utiliser pour controler la voiture. Je ne vais pas rentrer dans le détail du code HTML mais voici à quoi le site devrais ressembler :

Ensuite on va donner les instruction au serveur si on lui donne la requête «/control », qui ressort lorsqu'une direction est donnée au joystick du site internet.

```
server.on("/control", HTTP_GET, [](AsyncWebServerRequest *request) {
```

Tout d'abord on vérifie que les 2 paramètres dont on a besoin pour définir la direction de la voiture sont bien présents (coordonnées x et y du joystick)

```
if (request->hasParam("x") && request->hasParam("y")) {
```

Si ils existent, alors on stocke leurs valeurs dans des variables coté Arduino :

```
int x = request→getParam("x")→value().toInt();
int y = request->getParam("y")->value().toInt();
```

Et on donne ensuite ces valeurs à la fonction handle joystick

```
handleJoystick(x, y);
```

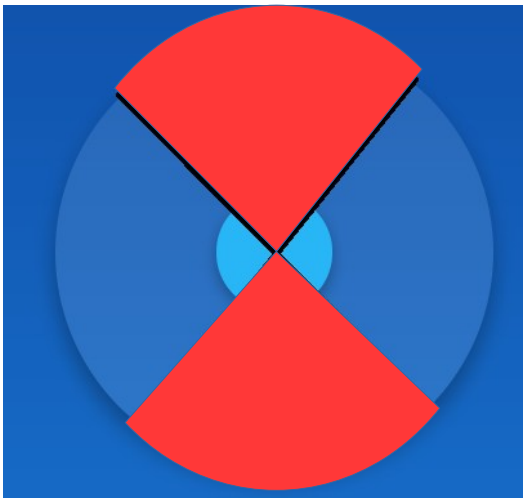
Cette fonction, définie par la suite dans le code, permet de définir la direction de la voiture en fonction de X et Y. :

```
void handleJoystick(int x, int y) {
```

Tout d'abord on défini la zone morte du joystick, en donnant comme condition que si les variables sont suffisamment faible (ici en dessous de 10) alors on arrête les moteurs.

```
if (abs(x) < 10 && abs(y) < 10) {
stopMotors();
return;
 }
```

Ensuite on prend les valeur avec la valeurs absolue de Y supérieur à celles de X soit la partie rouge du joystick :



Et on va ensuite différencier la partie haute (y positif) qui fais aller vers l'avant :

```
if (y > 0) {

    moveForward();
  }
```

Avec la fonction moveForward() définie plus bas comme suit :

```
void moveForward() {

   digitalWrite(IN1, HIGH);
   digitalWrite(IN2, LOW);
   digitalWrite(IN3, HIGH);
   digitalWrite(IN4, LOW);
}
```

Pour comprendre cette partie, il faut compendre comme on dirige les roues :

- IN1 et IN2 contrôlent la roue de gauche.
- IN3 et IN4 contrôlent la roue de droite.
- IN1 et IN3 font tourner leurs roues vers l'avant
- IN2 et IN4 font tourner leurs roues vers l'arrière

Il est donc logique que IN1 et IN3 soit activer pour aller vers l'avant des 2 cotés.

Et la partie basse (y négatif) qui devrais aller vers l'arrière

```
else {

     moveBackward();
   }
 }
```

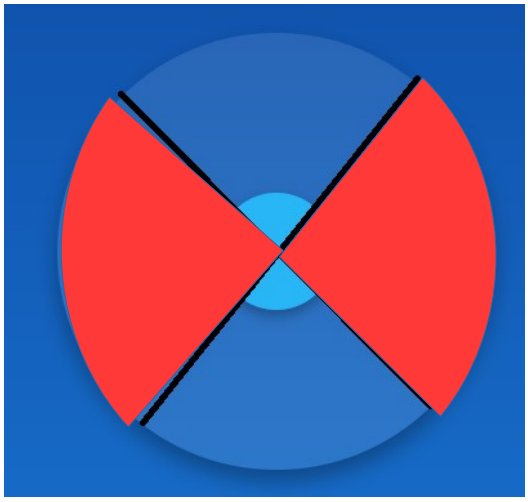
Avec la fonction moveBackward() définie plus haut comme suit :

```
void moveBackward() {

  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
}
```

Qui est l'exacte inverse de la fonction moveForward , ce qui est logique au niveau des sens de rotation de roue.

Si la condition n'est pas respéctées donc que l'on est dans l'autre partie du joystick (ici en rouge) :



Alors on veut tourner à droite ou à gauche, et pour les différencier on regarde x .
Si x est positif alors on est à droite du joystick on doit donc tourner à droite :

```
 else {

   if (x > 0) {
    turnRight(); }
```

Avec turnRight() qui est défini comme cela :

```
void turnRight() {

  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
}
```

Contrairement au fait de tourner vers l'avant ou l'arrière, afin de tourner on doit avoir des roues qui tourne dans des sens opposé.

La roue qui tourne vers l'arrière devant être celle du coté du où l'on souhaite tourner.

D'où le fait pour tourner à droite de faire tourner la roue de droite vers l'arrière (IN4 activé) et vers l'avant à gauche (IN1 activé)

Et sinon on tourne à gauche :

```
else {

    turnLeft();
   }
  }
}
```

Avec la fonction turnLeft définie comme suit :

```
void turnLeft() {
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
}
```

Qui est l'inverse de turnRight avec la roue gauche qui tourne vers l'arrière cette fois-ci.

Enfin pour terminer les instructions du serveur on renvoie un message de succès si on a bien eu une paire (x,y) à traiter et un message d'erreur sinon .

```
request->send(200, "text/plain", "OK");

   } else {
     request->send(400, "text/plain", "Paramètres manquants");
   }
 });
```

Et enfin pour terminer le void setup on lance notre serveur :

```
 server.begin();

}
```

On passe ensuite au void loop :

```
void loop() {
```

Où l'on commence par vérifier la distance au sol avec la fonction checkDistance() :

```
checkDistance();
```

Définie dans la suite du code comme cela :

```
void checkDistance() {
```

Tout d'abord, on stocke l'instant présent grâce à la fonction millis.

```
unsigned long currentMillis = millis();
```

Et on compare cela à la variable stocké pour la dernière mesure afin de savoir si l'écart entre les 2 est supéreur ou égal à l'intervalle défini plus haut entre chaque mesure.

```
if (currentMillis - lastMeasurement >= MEASURE_INTERVAL) {
```

Si c'est le cas alors on redéfini la variable qui stocke la dernière mesure, et on prend la distance au sol avec la fonction measureDistance() :

```
lastMeasurement = currentMillis;
float distance = measureDistance();
```

La fonction measureDistance() étant elle-même définie à part :

```
float measureDistance() {
```

Afin de faire fonctionner le capteur il faut arrêter le courant dans le pin trigger pour 2 microseconde, le rallumer pendant 10 microseconde puis l'éteindre à nouveau. On va donc exactement faire cela :

```
digitalWrite(TRIGGER_PIN, LOW);
delayMicroseconds(2);
digitalWrite(TRIGGER_PIN, HIGH);
delayMicroseconds(10);
digitalWrite(TRIGGER_PIN, LOW);
```

Avec la fonction delayMicrosecond qui permet d'exprimer les délais en microseconde et pas milliseconde comme les délais standards.

On calcule ensuite le temps que met le son à revenir en utilisant la fonction pulsIn qui donne le temps pendant lequel l'information données est vraie (à savoir ici combien de temps le pin echo met à en plus emettre de courant en microseconde)

```
long duration = pulseIn(ECHO_PIN, HIGH);
```

On va ensuite transformer ce temps en une distance connaissant la vitesse des ultrasons et divisant la distance pas 2 car les ultrasons doivent faire l'aller-retour jusqu'à l'objet.

```
float distance = duration * 0.034 / 2;
```

Et enfin on renvoie en sortie de fonction la distance à l'objet :

```
return distance;

}
```

Une fois revenu avec la bonne distance dans la fonction checkDistance. On met une condition sur la distance (positive et supérieure à la distance voulue)

```
if (distance > 0 && distance > MAX_DISTANCE) {
```

Si la condition est vérifiée alors on arrête les moteurs et on recule du bord duquel on pourrais tomber avant de s'arrêter à nouveau :

```
stopMotors();

    delay(50);
    moveBackward();
    delay(200);
    stopMotors();
  }
}
```

Et enfin on retourne dans le void loop() ou on demande au DNS de prendre en compte la prochaine requête afin d'éviter que le site ne fonctionne plus :

```
 dnsServer.processNextRequest();
}
```

4) Code Brut à recopié :

```
#include <ESP8266WiFi.h>
#include <ESPAsyncWebServer.h>
#include <DNSServer.h>


// Définition des broches
#define IN1 D1 // GPIO5
#define IN2 D2 // GPIO4
#define IN3 D5 // GPIO14
#define IN4 D6 // GPIO12


// Broches pour le HC-SR04
#define TRIGGER_PIN D7 // GPIO13
#define ECHO_PIN D8    // GPIO15

// Constantes pour le HC-SR04
const float MAX_DISTANCE = 5.0; // Distance maximale en cm avant arrêt
unsigned long lastMeasurement = 0;
const unsigned long MEASURE_INTERVAL = 10; // Intervalle de mesure en ms

// Configuration réseau
const char* ssid = "ESP-Robot"; // Nom du réseau Wi-Fi à modifier avec le MDP.
const char* password = "";     // Pas de mot de passe (réseau ouvert)

IPAddress apIP(192, 168, 4, 1); // AP local IP
DNSServer dnsServer;

// Serveur Web Asynchrone
AsyncWebServer server(80);

void setup() {
  // Configuration des broches
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);


  // Configuration du HC-SR04
  pinMode(TRIGGER_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);

  WiFi.softAP(ssid, password);
  WiFi.softAPConfig(apIP, IPAddress(192, 168, 4, 1), IPAddress(255, 255, 255, 0));
  dnsServer.start(53, "*", apIP);///Modifier le nom du site si voulus
  // Arrêter les moteurs au démarrage
```

```
  stopMotors();

  // Démarrer le Wi-Fi en mode point d'accès
  Serial.begin(115200);
  Serial.println();
  Serial.println("Point d'accès Wi-Fi créé");
  Serial.println("Adresse IP : " + WiFi.softAPIP().toString());

  // Page Web principale
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send(200, "text/html", getHTMLPage());
  });

  // Nouvelle route pour le contrôle joystick
  server.on("/control", HTTP_GET, [](AsyncWebServerRequest *request) {
    if (request->hasParam("x") && request->hasParam("y")) {
      int x = request->getParam("x")->value().toInt();
      int y = request->getParam("y")->value().toInt();
      handleJoystick(x, y);
      request->send(200, "text/plain", "OK");
    } else {
      request->send(400, "text/plain", "Paramètres manquants");
    }
  });

  // Démarrer le serveur
  server.begin();
}

void loop() {
  // Vérifier régulièrement la distance
  checkDistance();
  dnsServer.processNextRequest();
}


// Fonction pour mesurer la distance avec le HC-SR04
float measureDistance() {
  digitalWrite(TRIGGER_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIGGER_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIGGER_PIN, LOW);

  long duration = pulseIn(ECHO_PIN, HIGH);
  float distance = duration * 0.034 / 2; // Conversion en cm

  Serial.print("Distance mesurée: ");
```

```
  Serial.print(distance);
  Serial.println(" cm");

  return distance;
}

// Fonctions pour contrôler les moteurs
void moveForward() {
   Serial.println("Moteurs : Avancer");
   digitalWrite(IN1, HIGH);
   digitalWrite(IN2, LOW);
   digitalWrite(IN3, HIGH);
   digitalWrite(IN4, LOW);
}

void moveBackward() {
  Serial.println("Moteurs : Reculer");
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
}

void turnLeft() {
  Serial.println("Moteurs : Gauche");
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
}

void turnRight() {
  Serial.println("Moteurs : Droite");
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
}

void stopMotors() {
  Serial.println("Moteurs : Stop");
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
}
void handleJoystick(int x, int y) {
  // Zone morte pour éviter les mouvements involontaires
```

```
  if (abs(x) < 10 && abs(y) < 10) {
    stopMotors();
    return;
  }

  // Avancer/reculer
  if (abs(y) > abs(x)) {
    if (y > 0) {
      moveForward();
    } else {
      moveBackward();
    }
  }
  // Tourner gauche/droite
  else {
    if (x > 0) {
      turnRight();
    } else {
      turnLeft();
    }
  }
}

// Fonction pour vérifier la distance et arrêter les moteurs si nécessaire
void checkDistance() {
  unsigned long currentMillis = millis();
  if (currentMillis - lastMeasurement >= MEASURE_INTERVAL) {
    lastMeasurement = currentMillis;
    float distance = measureDistance();

    // Si la distance est superieru à la distance maximale et qu'un obstacle est détecté
    if (distance > 0 && distance > MAX_DISTANCE) {
      Serial.println("Obstacle détecté ! Arrêt des moteurs.");
      stopMotors();
      delay(50);
      moveBackward();
      delay(200);
      stopMotors();
    }
  }
}

// Fonction pour générer la page HTML avec joystick
String getHTMLPage() {
  return R"rawliteral(
<!DOCTYPE html>
<html lang="en">
<head>
```

```
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Robot Control</title>
<style>
  body {
    font-family: Arial, sans-serif;
    background: linear-gradient(to bottom, #0d47a1, #1976d2);
    color: white;
    text-align: center;
    margin: 0;
    padding: 0;
    height: 100vh;
    display: flex;
    flex-direction: column;
    justify-content: center;
  }

  h1 {
    margin-top: 0;
    font-size: 2.5em;
  }

  .joystick-container {
    width: 300px;
    height: 300px;
    margin: 20px auto;
    position: relative;
    background-color: rgba(255, 255, 255, 0.1);
    border-radius: 50%;
    box-shadow: 0 8px 15px rgba(0, 0, 0, 0.3);
  }

  .joystick {
    width: 80px;
    height: 80px;
    background-color: #29b6f6;
    border-radius: 50%;
    position: absolute;
    top: 110px;
    left: 110px;
    touch-action: none;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.3);
    transition: background 0.3s;
  }

  .joystick:active {
    background-color: #0288d1;
  }
```

```
    .info-panel {
      margin-top: 20px;
      color: #bbdefb;
      font-size: 0.9em;
    }

    .coordinates {
      margin-top: 10px;
      font-size: 1.2em;
      color: white;
    }
  </style>
</head>
<body>
  <h1>Contrôle du Robot ESP8266</h1>

  <div class="joystick-container" id="joystickArea">
    <div class="joystick" id="joystick"></div>
  </div>

  <div class="coordinates">
    X: <span id="xCoord">0</span> | Y: <span id="yCoord">0</span>
  </div>

  <div class="info-panel">
    Sécurité active: Le robot s'arrête automatiquement si un obstacle est détecté à moins de 5 cm.
  </div>

  <script>
    const joystick = document.getElementById('joystick');
    const joystickArea = document.getElementById('joystickArea');
    const xCoord = document.getElementById('xCoord');
    const yCoord = document.getElementById('yCoord');

    let isDragging = false;
    const maxDistance = 100;
    const center = { x: 150, y: 150 };
    const joystickSize = 40;

    // Position initiale
    joystick.style.left = (center.x - joystickSize) + 'px';
    joystick.style.top = (center.y - joystickSize) + 'px';

    function updatePosition(clientX, clientY) {
      const rect = joystickArea.getBoundingClientRect();
      let x = clientX - rect.left - center.x;
      let y = clientY - rect.top - center.y;
```

```
  // Limiter le joystick au cercle
  const distance = Math.sqrt(x*x + y*y);
  if (distance > maxDistance) {
    x = x * maxDistance / distance;
    y = y * maxDistance / distance;
  }

  // Mettre à jour la position visuelle
  joystick.style.left = (center.x + x - joystickSize) + 'px';
  joystick.style.top = (center.y + y - joystickSize) + 'px';

  // Mettre à jour les coordonnées affichées
  const scaledX = Math.round(x / maxDistance * 100);
  const scaledY = Math.round(y / maxDistance * 100);
  xCoord.textContent = scaledX;
  yCoord.textContent = -scaledY;

  // Envoyer les commandes au robot
  sendJoystickCommand(scaledX, -scaledY);
}

function sendJoystickCommand(x, y) {
  fetch(`/control?x=${x}&y=${y}`)
    .catch(err => console.error('Erreur:', err));
}

function resetPosition() {
  joystick.style.left = (center.x - joystickSize) + 'px';
  joystick.style.top = (center.y - joystickSize) + 'px';
  xCoord.textContent = '0';
  yCoord.textContent = '0';
  sendJoystickCommand(0, 0);
}

// Événements tactiles
joystick.addEventListener('touchstart', (e) => {
  e.preventDefault();
  isDragging = true;
});

document.addEventListener('touchmove', (e) => {
  if (isDragging) {
    e.preventDefault();
    updatePosition(e.touches[0].clientX, e.touches[0].clientY);
  }
}, { passive: false });
```

```
  document.addEventListener('touchend', (e) => {
    if (isDragging) {
      isDragging = false;
      resetPosition();
    }
  });

  // Événements souris
  joystick.addEventListener('mousedown', (e) => {
    e.preventDefault();
    isDragging = true;
  });

  document.addEventListener('mousemove', (e) => {
    if (isDragging) {
      e.preventDefault();
      updatePosition(e.clientX, e.clientY);
    }
  });

  document.addEventListener('mouseup', (e) => {
    if (isDragging) {
      isDragging = false;
      resetPosition();
    }
  });
 </script>
</body>
</html>
)rawliteral";
}
```