```
// --- DÉFINITION DES BROCHES (PINS) ---

// Pins pour le contrôle des moteurs (via Driver L298N)
#define In1 2       // Direction moteur Droit 1
#define In2 3       // Direction moteur Droit 2
#define EnA 5       // Vitesse (PWM) moteur Droit
#define In3 4       // Direction moteur Gauche 1
#define In4 7       // Direction moteur Gauche 2
#define EnB 6       // Vitesse (PWM) moteur Gauche

// Pins pour les capteurs Ultrasons (Définis mais non utilisés dans la boucle principale ici)
#define TrigM 13
#define EchoM A1
#define EchoG 12
#define EchoD A3
#define TrigD A0
#define TrigG 11

// Pins pour les capteurs Infrarouges (Utilisés pour la mesure de distance dans ce code)
// "Vir" signifie probablement "Vision InfraRouge"
#define VirM A7     // Capteur IR Milieu
#define VirG A2     // Capteur IR Gauche
#define VirD A6     // Capteur IR Droit

// --- CONSTANTES ET PARAMÈTRES ---

// États du système
#define mAuto 0     // Mode Automatique
#define mManu 1     // Mode Manuel (Télécommandé)

// Identifiants pour la fonction de pilotage moteur
#define Left 0      // Moteur Gauche
#define Right 1     // Moteur Droit
#define Forw 1      // Marche Avant (Forward)
#define Backw -1    // Marche Arrière (Backward)
#define Stop 0      // Arrêt

// Pins d'entrée du récepteur Radio-commande (RC)
#define v1 8        // Canal 1 : Direction (Molette)
#define v2 9        // Canal 2 : Vitesse (Gâchette)
#define v3 10       // Canal 3 : Sélecteur de mode (Bouton)

// Bibliothèques pour l'écran LCD I2C
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Initialisation de l'objet LCD (Adresse 0x27, 16 colonnes, 2 lignes)
LiquidCrystal_I2C lcd(0x27, 16, 2);
```

```
// --- VARIABLES GLOBALES ---

unsigned long mol;  // Stocke la valeur brute du canal direction (µs)
unsigned long gach; // Stocke la valeur brute du canal vitesse (µs)
unsigned long btn;  // Stocke la valeur brute du canal mode (µs)

int vitesse, SensD , SensG;         // Vitesse globale et sens de chaque moteur
int coeffG, coeffD, PowerD, PowerG; // Coefficients de virage et puissance finale envoyée aux moteurs
float MD, MG;                       // Variables intermédiaires (peu utilisées ici)
bool mode;                          // Stocke le mode actuel (Auto ou Manu)

// --- INITIALISATION (SETUP) ---
void setup() {
  // Configuration des pins moteurs en SORTIE
  pinMode(In1, OUTPUT);
  pinMode(In2, OUTPUT);
  pinMode(EnA, OUTPUT);
  pinMode(In3, OUTPUT);
  pinMode(In4, OUTPUT);
  pinMode(EnB, OUTPUT);

  // Configuration des pins récepteur RC en ENTRÉE
  pinMode(v1, INPUT);
  pinMode(v2, INPUT);
  pinMode(v3, INPUT);

  Serial.begin(9600); // Démarrage de la communication série pour le débogage

  // Configuration des pins capteurs (Même si les IR utilisent analogRead qui ne nécessite pas forcément pinMode)
  pinMode(TrigM, OUTPUT);
  pinMode(TrigG, OUTPUT);
  pinMode(TrigD, OUTPUT);
  pinMode(EchoM, INPUT);
  pinMode(EchoG, INPUT);
  pinMode(EchoD, INPUT);
  pinMode(VirM, INPUT);
  pinMode(VirD, INPUT);
  pinMode(VirG, INPUT);

  // Démarrage de l'écran LCD
  lcd.backlight(); // Allume le rétro-éclairage
  lcd.begin();     // Initialise l'affichage
}

// --- BOUCLE PRINCIPALE (LOOP) ---
```

```
void loop() {
  // 1. LECTURE DES SIGNAUX RADIO
  // pulseIn lit la durée de l'impulsion PWM en microsecondes
  gach = pulseIn(v2, 30000); // Lecture gaz (Timeout 30ms)
  btn = pulseIn(v3, HIGH);   // Lecture bouton mode
  mol = pulseIn(v1, HIGH);   // Lecture direction

  // 2. NETTOYAGE DES VALEURS RC (SATURATION)
  // On force les valeurs à rester dans la plage standard 1000-2000 µs
  // pour éviter les bugs si le signal est perdu ou parasité.
  if (mol > 2000) { mol = 2000; }
  if (mol < 1000) { mol = 1000; }

  if (gach < 1000) { gach = 1000; }
  if (gach > 2000) { gach = 2000; }

  if (btn < 1000) { btn = 1000; }

  // 3. DÉTECTION DU MODE (AUTO ou MANU)
  // Bascule selon la position du commutateur sur la télécommande (seuil à 1500)
  if (btn > 1500) {
    mode = mAuto;
  }
  if (btn < 1500) {
    mode = mManu;
  }

  // --- LOGIQUE MODE MANUEL ---
  if (mode == mManu) {

    // A. Gestion de l'Avance / Recul (Gâchette)
    // Si gâchette > 1550 : Marche Avant
    if (gach > 1550) {
      vitesse = map(gach, 1550, 2000, 0, 100); // Convertit 1550-2000 en 0-100%
      SensG = Forw;
      SensD = Forw;
    }
    // Si gâchette < 1450 : Marche Arrière
    if (gach < 1450) {
      vitesse = map(gach, 1450, 1000, 0, 100); // Convertit 1450-1000 en 0-100%
      SensG = Backw;
      SensD = Backw;
    }
    // Zone morte centrale (entre 1450 et 1550) : Arrêt
    if ((gach > 1450) && (gach < 1550)) {
      vitesse = 0;
      // lcd.print(" Mode Manu "); // (Code commenté pour l'affichage)
    }
```

```
  // B. Gestion de la Direction (Molette) - Mixage différentiel
  // Tourner à Gauche ou Droite en réduisant la vitesse d'un côté

  // Si molette > 1550 (Vers un côté)
  if (mol > 1550) {
    coeffG = 255; // Gauche à fond
    coeffD = map(mol, 1550, 2000, 255, 0); // Droit ralentit progressivement
  }
  // Si molette < 1450 (Vers l'autre côté)
  if (mol < 1450) {
    coeffD = 255; // Droit à fond
    coeffG = map(mol, 1450, 1000, 255, 0); // Gauche ralentit progressivement
  }
  // Zone morte direction (Tout droit)
  if (mol > 1450 && mol < 1550) {
    coeffD = 255;
    coeffG = 255;
  }

  // C. Calcul de la puissance finale (Mixage Vitesse * Direction)
  PowerD = (vitesse * coeffD) / 100;
  PowerG = (vitesse * coeffG) / 100;

  // Sécurité : On s'assure que le PWM ne dépasse pas 255 (max sur Arduino 8 bits)
  if (PowerD > 255) {
    PowerD = 255;
    MD = PowerD * 2.55; // Calcul intermédiaire (non utilisé ensuite)
  }
  if (PowerG > 255) {
    PowerG = 255;
    MG = PowerG * 2.55;
  }

  // Envoi des commandes aux moteurs via la fonction dédiée
  cmd_motor(Left, SensG, PowerG);
  cmd_motor(Right, SensD, PowerD);

}
// --- LOGIQUE MODE AUTOMATIQUE (Evitement d'obstacles) ---
else {
  // Lecture des distances via les fonctions de conversion (voir plus bas)
  int dM = Millieu();
  int dG = Gauche();
  int dD = Droite();
  int d = 35; // Seuil de distance de sécurité (35 cm)

  // Timer non bloquant pour exécuter la logique toutes les 300ms
```

```
  static unsigned long previousmillis = 0;
  unsigned long currentmillis = millis();

  if (currentmillis - previousmillis >= 300) {
    previousmillis = currentmillis;

    // Scénario 1 : Voie libre partout (Tout > 35cm) -> Avancer
    if((dG>d)&&(dM>d)&&(dD>d)){
      SensG = Forw;
      SensD = Forw;
      PowerG = 90;
      PowerD = 90;
    }

    // Scénario 2 : Obstacle à Droite (dD < d), le reste libre -> Tourner légèrement
(ajustement puissance)
    if((dG>d)&&(dM>d)&&(dD<d)){
      SensG = Forw;
      PowerG = 78; // Gauche plus fort
      PowerD = 40; // Droite moins fort
    }

    // Scénario 3 : Obstacle au Milieu, côtés libres
    if((dG>d)&&(dM<d)&&(dD>d)){
      // Choix aléatoire pour contourner par la gauche ou la droite
      if(random(0,2) == 0){
        SensG = Forw;
        PowerG = 78;
        PowerD = 40;
      }
      else{
        SensD = Forw;
        PowerD = 78;
        PowerG = 40;
      }
    }

    // Scénario 4 : Obstacle Milieu et Droite -> Tourner
    if((dG>d)&&(dM<d)&&(dD<d)){
      SensG = Forw;
      PowerG = 78;
      PowerD = 40;
    }

    // Scénario 5 : Obstacle Gauche -> Tourner vers la Droite
    if((dG<d)&&(dM>d)&&(dD>d)){
      SensD = Forw;
      PowerD = 78;
```

```
      PowerG = 40;
    }

    // Scénario 6 : Obstacle Gauche et Droite, Milieu libre (Couloir ?) -> Avancer prudent
    if((dG<d)&&(dM>d)&&(dD<d)){
      SensG = Forw;
      SensD = Forw;
      PowerD = 90;
      PowerG = 78;
    }

    // Scénario 7 : Obstacle Gauche et Milieu -> Tourner
    if((dG<d)&&(dM<d)&&(dD>d)){
      SensD = Forw;
      PowerD = 78;
      PowerG = 40;
    }

    // Scénario 8 : Tout est bloqué (Cul de sac) -> Reculer
    if((dG<d)&&(dM<d)&&(dD<d)){
      SensD = Backw;
      SensG = Backw;
      PowerD = 120;
      PowerG = 0; // Note : Un moteur à 0 fera pivoter le robot en reculant
      delay(750); // Petite pause bloquante pour effectuer la manœuvre
    }

    // Application des commandes calculées ci-dessus
    cmd_motor(Left, SensG, PowerG);
    cmd_motor(Right, SensD, PowerD);
   }
  }
}

// --- FONCTIONS DE MESURE DE DISTANCE (Capteurs IR) ---
// Ces fonctions lisent la valeur analogique et appliquent une formule mathématique
// pour linéariser la réponse du capteur IR (Tension -> Distance en cm)

float Millieu() {
  int im = analogRead(VirM); // Lecture analogique
  // Formule spécifique aux capteurs Sharp IR : Distance = Constante * (Valeur ^ -1.26)
  float N0 = 21767.668 * (pow(im, -1.261246510));
  return N0; // Retourne la distance
}

float Droite() {
  int im = analogRead(VirD);
  float N1 = 21767.668 * (pow(im, -1.261246510));
```

```
  return N1;
}

float Gauche() {
  int im = analogRead(VirG);
  float N2 = 21767.668 * (pow(im, -1.261246510));
  return N2;
}

// --- FONCTION DE COMMANDE MOTEUR (Abstraction Hardware) ---
// Simplifie le code principal en gérant les états des pins du driver L298N
void cmd_motor (int motor, int direction, int power) {
  // Serial.println("power" + String(power)); // Debug

  // Pilotage Moteur DROIT
  if (motor == Right) {
    if (direction == Forw) {
      digitalWrite(In1, LOW);
      digitalWrite(In2, HIGH);
      analogWrite(EnA, power); // Vitesse via PWM
    }
    if (direction == Backw) {
      digitalWrite(In1, HIGH);
      digitalWrite(In2, LOW);
      analogWrite(EnA, power);
    }
    if (direction == Stop) {
      analogWrite(EnA, 0); // Coupe l'alimentation
    }
  }

  // Pilotage Moteur GAUCHE
  if (motor == Left) {
    if (direction == Forw) {
      digitalWrite(In3, HIGH);
      digitalWrite(In4, LOW);
      analogWrite(EnB, power);
    }
    if (direction == Backw) {
      digitalWrite(In3, LOW);
      digitalWrite(In4, HIGH);
      analogWrite(EnB, power);
    }
    if (direction == Stop) {
      analogWrite(EnB, 0);
    }
  }
}
```