

Rapport de projet

Conception d'un robot mobile autonome

Adaptation du kit Sumo-BOT (Robo-Creator2) sur Arduino UNO

Évitement d'obstacles • Contrôle Bluetooth • RemoteXY

Document explicatif — processus de réalisation complet

Table des matières

Table des matières	2
1. Introduction et objectifs	3
Objectifs poursuivis	3
2. Liste du matériel utilisé.....	3
2.1 Éléments réutilisés du kit Robo-Creator2.....	3
2.2 Composants ajoutés.....	3
3. Pourquoi et comment passer de l'ATX2 à l'Arduino UNO	5
3.1 Choix du driver moteur	5
3.2 Pilotage différentiel à 2 canaux.....	5
4. Schémas et tableaux de câblage	5
4.1 Câblage moteurs (L298N)	5
4.2 Câblage des capteurs.....	6
4.3 Câblage du module Bluetooth HC-05	6
4.4 Récapitulatif global des pins Arduino UNO	6
5. Photos du montage réalisé.....	8
6. Démarche de test et de mise au point.....	11
7. Code source.....	12
7.1 Code de test moteur (validation du sens de rotation)	12
7.2 Code final — évitement d'obstacles (mode autonome)	13
7.3 Code final — contrôle Bluetooth (RemoteXY).....	15
8. Difficultés rencontrées et solutions apportées	17
9. Conclusion	18

1. Introduction et objectifs

Ce rapport documente la réalisation d'un robot mobile autonome à partir du kit pédagogique « Sumo-BOT » du Robo-Creator2 (Innovative Experiment Co., Ltd.). Le kit d'origine est conçu pour fonctionner avec la carte propriétaire ATX2 (ATmega644). L'objectif de ce projet a été de réutiliser entièrement la structure mécanique du kit (châssis, moteurs, roues) tout en remplaçant le contrôleur par un Arduino UNO, plus accessible et largement documenté.

Le projet final ne reproduit pas le comportement « sumo » d'origine (combat sur un ring avec détection de ligne), mais a évolué vers un robot d'évitement d'obstacles, pilotable également à distance via Bluetooth grâce à l'application RemoteXY.

Objectifs poursuivis

- Remplacer la carte ATX2 par un Arduino UNO
- Piloter les 4 moteurs DC du châssis via un driver moteur externe
- Ajouter un capteur à ultrasons pour la détection d'obstacles
- Ajouter un capteur infrarouge en complément de détection
- Permettre un contrôle filaire (PC) et un contrôle sans fil (Bluetooth / smartphone)

2. Liste du matériel utilisé

2.1 Éléments réutilisés du kit Robo-Creator2

- Châssis acrylique complet (pièces Chassis A, B, C, D)
- 4 moteurs DC gearbox BO-1 (ratio 87:1)
- 4 roues avec jantes et hubs
- Support de batterie 6×AA avec câble d'alimentation
- Scoop avant métallique, visserie et entretoises
- 2 capteurs infrarouges ZX-03 (montés sous le châssis, non utilisés dans la version finale)

2.2 Composants ajoutés

Composant	Rôle dans le projet
Arduino UNO	Remplace la carte ATX2 — contrôleur principal
Module L298N (driver moteur)	Pilotage des 4 moteurs DC (en paires gauche / droite)
Capteur ultrason HC-SR04	Détection de la distance à un obstacle
Module infrarouge (avec potentiomètre de seuil)	Détection complémentaire de proximité
Module Bluetooth HC-05	Communication sans fil avec un smartphone
Résistances 1 kΩ (x2, montées en série)	Diviseur de tension pour protéger l'entrée RX du HC-05
Câbles Dupont (M-M, M-F, F-F)	Câblage de l'ensemble du montage

Composant	Rôle dans le projet
Application RemoteXY	Interface de contrôle à distance (joystick) sur smartphone

3. Pourquoi et comment passer de l'ATX2 à l'Arduino UNO

La carte ATX2 d'origine intègre un driver moteur 6 canaux, un convertisseur analogique-numérique dédié et une bibliothèque logicielle de haut niveau (fonctions `fd()`, `bk()`, `tl()`, `tr()`, `sl()`, `sr()`...). L'Arduino UNO ne possède aucune de ces fonctionnalités nativement : il a fallu ajouter un driver moteur externe (L298N) et réécrire l'ensemble de la logique de pilotage et de lecture de capteurs.

3.1 Choix du driver moteur

Deux composants ont été envisagés pour piloter les moteurs : le L298N et le L293D. Le L293D, bien que plus compact, est limité à environ 600 mA par canal, ce qui est insuffisant pour des moteurs DC pouvant solliciter un courant élevé en cas de blocage (poussée contre un obstacle). Le module L298N standard (avec dissipateur thermique intégré) a donc été retenu, pour sa meilleure tenue en courant et en température.

3.2 Pilotage différentiel à 2 canaux

Le L298N possède deux canaux indépendants (A et B). Plutôt que de piloter les 4 moteurs séparément comme le permettait l'ATX2, les moteurs ont été câblés par paires en parallèle : les deux moteurs du côté gauche sur le canal A, les deux moteurs du côté droit sur le canal B. Cette configuration, dite « pilotage différentiel », permet de réaliser tous les mouvements nécessaires (avancer, reculer, tourner, pivoter sur place) avec un seul module L298N.

4. Schémas et tableaux de câblage

4.1 Câblage moteurs (L298N)

Borne L298N	Connexion
OUT1	Fils (+) des 2 moteurs gauche (avant-gauche + arrière-gauche)
OUT2	Fils (-) des 2 moteurs gauche
OUT3	Fils (+) des 2 moteurs droite (avant-droit + arrière-droit)
OUT4	Fils (-) des 2 moteurs droite
IN1	Arduino D4
IN2	Arduino D5
IN3	Arduino D7
IN4	Arduino D8
ENA / ENB	Jumpers laissés en place (vitesse fixe, sans contrôle PWM)
+12V / VM	Borne (+) du pack batterie
GND	Borne (-) du pack batterie, commun avec Arduino

4.2 Câblage des capteurs

Capteur	Pin	Arduino
HC-SR04	VCC	5V
HC-SR04	TRIG	D12
HC-SR04	ECHO	D13
HC-SR04	GND	GND
Module Infrarouge	VCC	5V
Module Infrarouge	GND	GND
Module Infrarouge	OUT	A0

4.3 Câblage du module Bluetooth HC-05

HC-05	Connexion
VCC	Arduino 5V
GND	Arduino GND
TX	Arduino D2 (direct, sans résistance)
RX	Arduino D6, via un diviseur de tension (2 résistances de 1 k Ω en série)

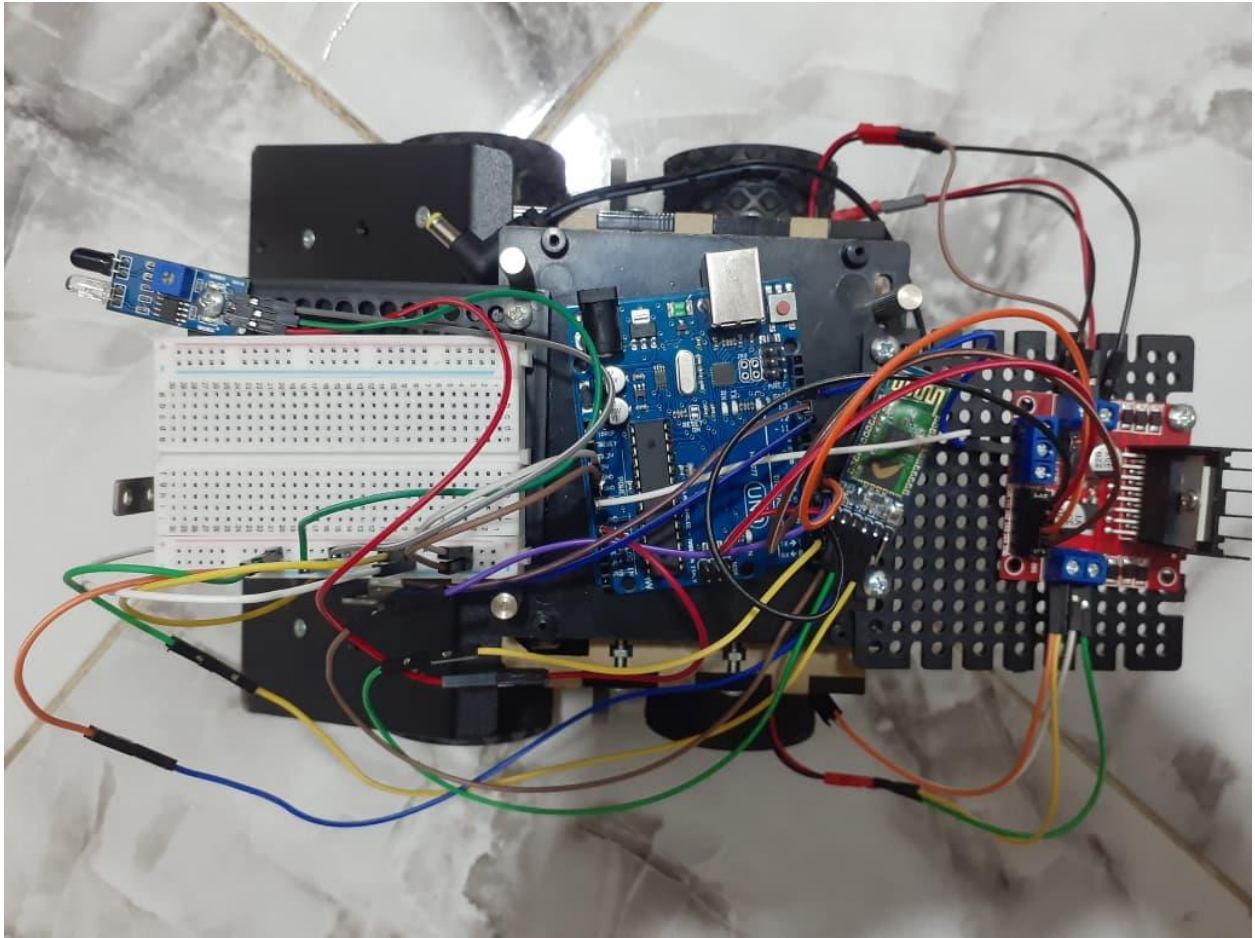
Le HC-05 fonctionne en logique 3,3 V alors que l'Arduino émet en 5 V sur ses sorties numériques. Un diviseur de tension a donc été ajouté uniquement sur la ligne RX du module (Arduino → HC-05), afin de ramener le signal à un niveau sûr (environ 2,5 V avec deux résistances égales de 1 k Ω), évitant ainsi d'endommager le module à terme. La ligne TX du HC-05 vers l'Arduino ne nécessite pas d'adaptation, le 3,3 V étant correctement interprété comme un niveau logique haut par l'Arduino.

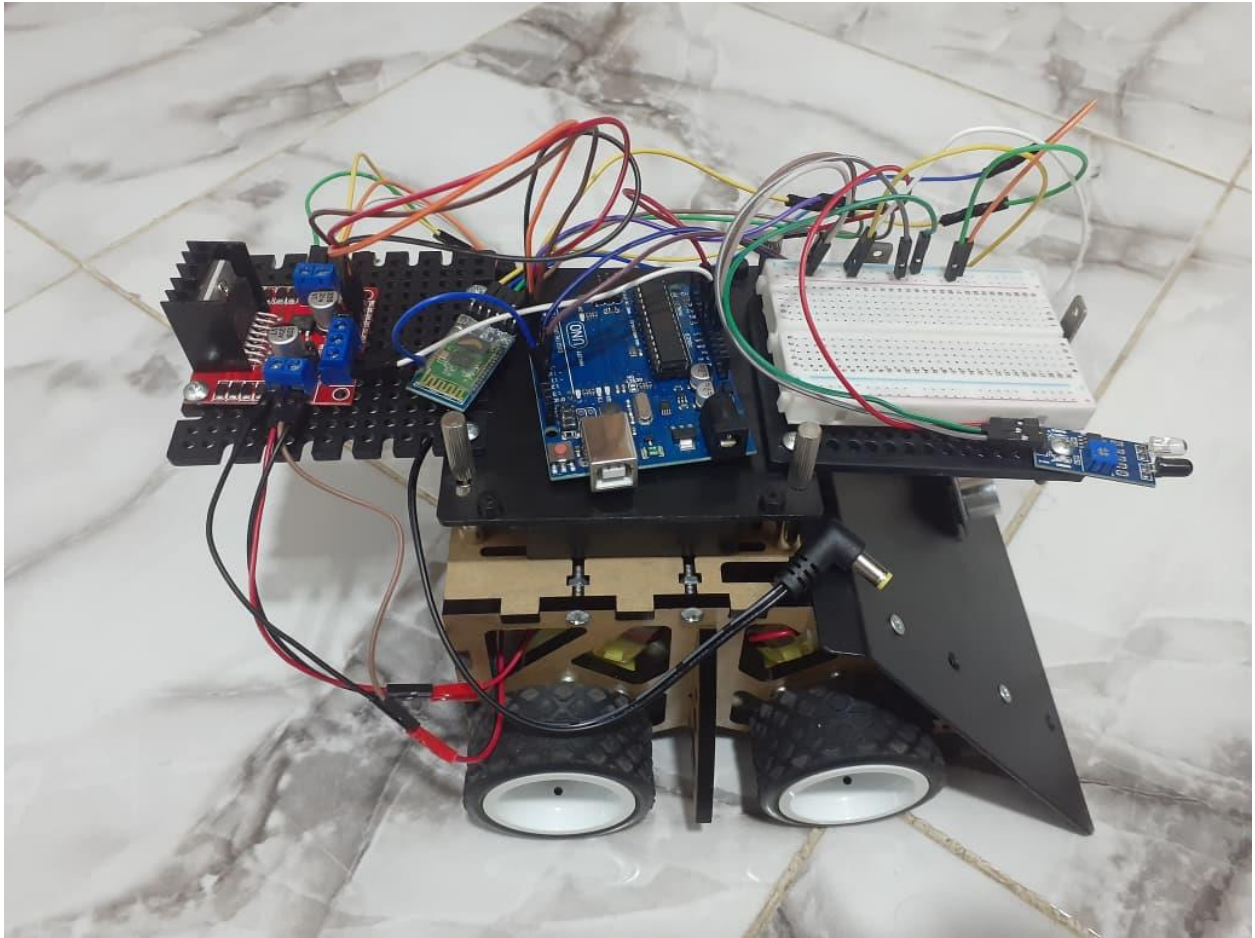
4.4 Récapitulatif global des pins Arduino UNO

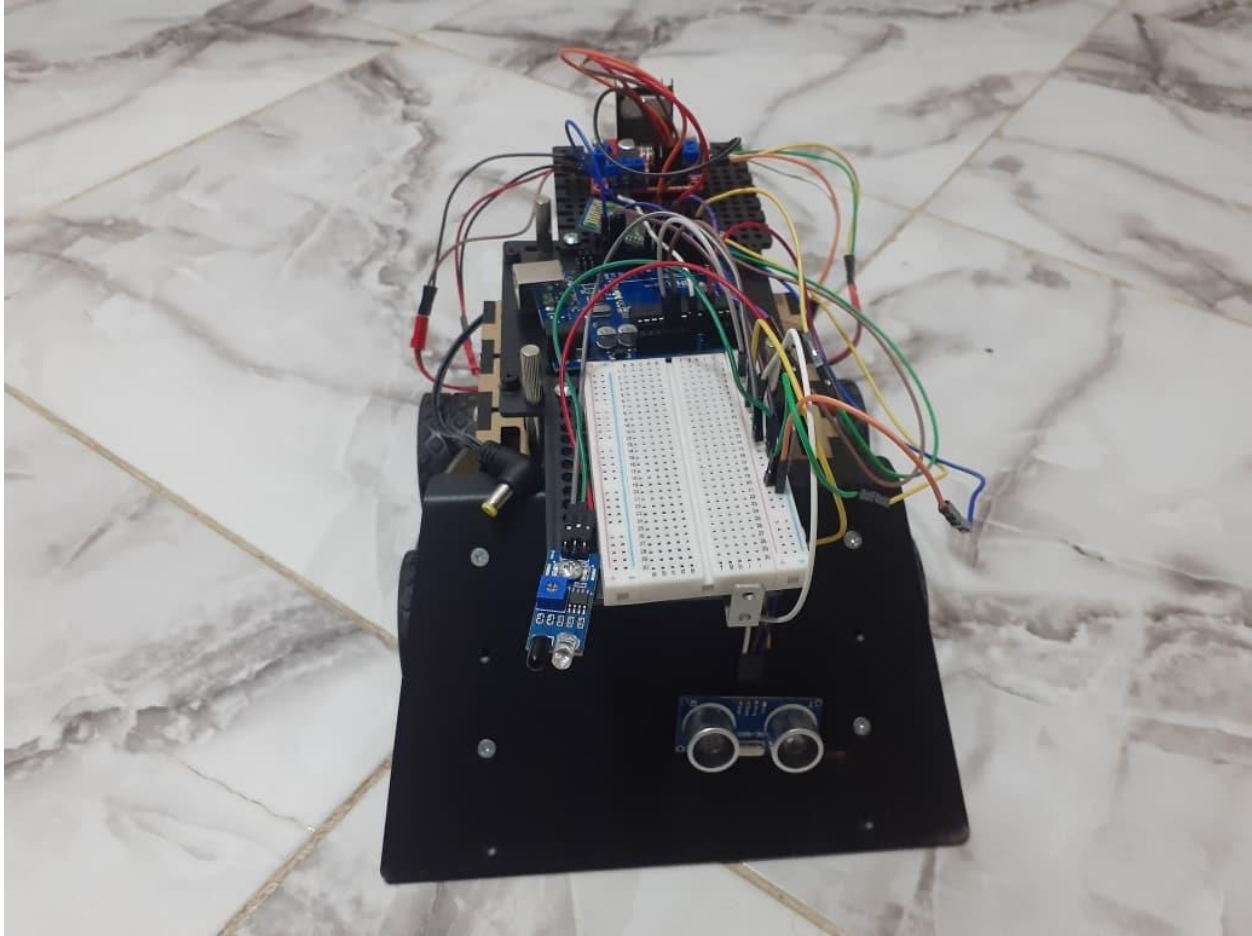
Pin Arduino	Connecté à
D2	HC-05 TX
D4	L298N IN1
D5	L298N IN2
D6	HC-05 RX (via diviseur de tension)
D7	L298N IN3
D8	L298N IN4
D12	HC-SR04 TRIG
D13	HC-SR04 ECHO
A0	Module infrarouge (OUT)

Pin Arduino	Connecté à
5V / GND	Alimentation commune de tous les modules

5. Photos du montage réalisé







6. Démarche de test et de mise au point

La mise au point du robot s'est faite par étapes successives, en isolant chaque sous-système avant de les combiner :

- Test individuel des 4 moteurs (rotation à vide, roues soulevées) pour vérifier le sens de rotation
- Détection et correction d'une inversion de polarité sur les moteurs du côté droit (fils OUT3/OUT4 inversés)
- Validation du pilotage différentiel : avancer, reculer, virages, pivot sur place
- Calibration du module infrarouge à l'aide du potentiomètre intégré
- Test de lecture du capteur à ultrasons (mesure de distance via le moniteur série)
- Réduction de la dépendance au capteur infrarouge dans la logique de décision, le HC-SR04 étant moins sensible aux conditions lumineuses ambiantes
- Intégration du module Bluetooth HC-05 et validation de la communication série
- Conception de l'interface de pilotage avec RemoteXY (joystick) et fusion avec le code de contrôle moteur
- Validation du fonctionnement autonome sur batterie, sans câble USB

Un point d'attention particulier a porté sur l'alimentation : le robot doit être alimenté à la fois en USB (lors des phases de programmation et de débogage) et par le pack de batteries (alimentation des moteurs, via la broche Vin de l'Arduino). Une fois le code validé et téléversé,

le câble USB n'est plus nécessaire : le programme reste stocké dans la mémoire de l'Arduino et le robot fonctionne de façon autonome sur batterie.

7. Code source

7.1 Code de test moteur (validation du sens de rotation)

Ce premier code ne sert qu'à valider le câblage des moteurs avant d'aller plus loin. Il ne fait appel à aucun capteur : il commande uniquement les 4 broches de direction du L298N (IN1 à IN4), une combinaison à la fois, avec des pauses permettant d'observer visuellement chaque mouvement.

```
#define IN1 4
#define IN2 5
#define IN3 7
#define IN4 8

void setup() {
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
}

void loop() {
  // Test côté GAUCHE seul
  digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
  delay(2000);
  digitalWrite(IN1, LOW); digitalWrite(IN2, LOW);
  delay(1000);

  // Test côté DROIT seul
  digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW);
  delay(2000);
  digitalWrite(IN3, LOW); digitalWrite(IN4, LOW);
  delay(1000);

  // Test AVANCER (les 2 côtés ensemble)
  digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW);
  delay(2000);
  digitalWrite(IN1, LOW); digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW); digitalWrite(IN4, LOW);
  delay(2000);
}
```

Explication ligne par ligne :

- `#define IN1 4` (et les 3 suivants) : associent un nom lisible (IN1, IN2, IN3, IN4) aux numéros de broches physiques de l'Arduino, pour rendre le code plus clair que de manipuler directement les chiffres 4, 5, 7, 8.
- `setup()` : exécuté une seule fois au démarrage. `pinMode(..., OUTPUT)` déclare chaque broche comme une sortie numérique, c'est-à-dire que l'Arduino va y imposer une tension (0V ou 5V) plutôt que d'y lire un signal.
- `loop()` : exécuté en boucle infinie. C'est ici que se déroule le test proprement dit.

- `digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);` : sur le canal A du L298N, mettre IN1 à l'état haut (5V) et IN2 à l'état bas (0V) fait tourner les moteurs du côté gauche dans un sens donné. Inverser ces deux états inverserait le sens de rotation.
- `delay(2000);` : pause de 2000 millisecondes (2 secondes), le temps d'observer la rotation des roues.
- `digitalWrite(IN1, LOW); digitalWrite(IN2, LOW);` : mettre les deux broches à l'état bas arrête le moteur (aucune différence de potentiel appliquée).
- Le même principe est répété pour IN3/IN4 (côté droit), puis pour les 4 broches combinées (test du mouvement global « avancer »).

C'est ce code qui a permis de détecter l'inversion de polarité sur le côté droit, corrigée ensuite en intervertissant physiquement les fils sur les bornes OUT3/OUT4 du L298N.

7.2 Code final — évitement d'obstacles (mode autonome)

Ce code combine le pilotage des moteurs avec la lecture de deux capteurs (HC-SR04 et module infrarouge) pour permettre au robot de se déplacer seul, sans intervention humaine, en évitant les obstacles qu'il rencontre.

```
// ===== Pins moteurs =====
#define IN1 4
#define IN2 5
#define IN3 7
#define IN4 8

// ===== Pins capteurs =====
#define TRIG 12
#define ECHO 13
#define IR_PIN A0

int seuilDistance = 20; // cm

void setup() {
  pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT);
  pinMode(TRIG, OUTPUT); pinMode(ECHO, INPUT);
  pinMode(IR_PIN, INPUT);
  Serial.begin(9600);
}

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

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

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

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

void arreter() {
  digitalWrite(IN1, LOW); digitalWrite(IN2, LOW);
}
```

```

    digitalWrite(IN3, LOW); digitalWrite(IN4, LOW);
}

long getDistance() {
    digitalWrite(TRIG, LOW); delayMicroseconds(2);
    digitalWrite(TRIG, HIGH); delayMicroseconds(10);
    digitalWrite(TRIG, LOW);
    long duration = pulseIn(ECHO, HIGH, 30000);
    if (duration == 0) return 999;
    return duration * 0.034 / 2;
}

void loop() {
    long distance = getDistance();
    int irDetecte = digitalRead(IR_PIN);

    bool obstacleProche = (distance < seuilDistance);
    bool obstacleTresProche = (distance < (seuilDistance - 5));

    // Le HC-SR04 reste prioritaire ; l'IR ne fait que confirmer
    bool obstacleDetecte = obstacleTresProche ||
        (obstacleProche && irDetecte == LOW);

    if (obstacleDetecte) {
        arreter(); delay(200);
        reculer(); delay(400);
        arreter(); delay(100);
        if (random(0, 2) == 0) tournerGauche();
        else tournerDroite();
        delay(500);
        arreter(); delay(100);
    } else {
        avancer();
    }
}

```

Explication des parties principales :

- Section « Pins » : reprend les mêmes définitions de broches moteurs que le code de test, et ajoute TRIG/ECHO (capteur à ultrasons) et IR_PIN (capteur infrarouge, branché en A0).
- seuilDistance = 20 : variable réglable qui fixe la distance (en centimètres) en dessous de laquelle un obstacle est considéré comme « proche ». Diminuer cette valeur rend le robot plus « téméraire » (il s'approche davantage avant de réagir), l'augmenter le rend plus prudent.
- Les fonctions avancer(), reculer(), tournerGauche(), tournerDroite() et arreter() encapsulent chacune une combinaison d'états HIGH/LOW sur IN1-IN4, validée lors des tests précédents. Elles rendent le reste du code plus lisible : on écrit avancer() plutôt que de répéter 4 lignes de digitalWrite à chaque fois.
- getDistance() : fonction qui mesure la distance avec le capteur à ultrasons. Elle envoie une brève impulsion de 10 microsecondes sur TRIG (ce qui déclenche l'émission d'un signal ultrasonore), puis mesure avec pulseIn() le temps que met l'écho à revenir sur ECHO. Ce temps est ensuite converti en distance grâce à la vitesse du son (formule : $\text{distance} = \text{durée} \times 0,034 / 2$, le $\div 2$ correspondant à l'aller-retour de l'onde). Le paramètre 30000 dans pulseIn() est un délai d'attente maximal (30 ms) : si aucun écho

n'est reçu (rien en face), la fonction retourne 0 et le code renvoie alors une distance conventionnelle de 999 cm (« rien détecté »).

- `obstacleProche` et `obstacleTresProche` : deux booléens qui comparent la distance mesurée au seuil, avec une marge de 5 cm entre les deux, pour distinguer une zone de « vigilance » d'une zone de « danger immédiat ».
- `obstacleDetecte = obstacleTresProche || (obstacleProche && irDetecte == LOW)` : c'est le cœur de la logique de décision. Un obstacle très proche déclenche toujours l'évitement (priorité sécurité). Dans la zone intermédiaire, l'évitement n'est déclenché que si le capteur infrarouge confirme également une détection — ceci a été mis en place pour réduire les faux positifs liés à la sensibilité du capteur infrarouge aux variations de lumière ambiante (voir section 8).
- Le bloc if (`obstacleDetecte`) { ... } décrit la manœuvre d'évitement : arrêt bref, recul, nouvel arrêt, puis rotation choisie aléatoirement (`random(0,2)` renvoie 0 ou 1) à gauche ou à droite, avant de reprendre la marche normale. Le choix aléatoire évite que le robot ne reste bloqué en boucle en tournant toujours du même côté face à un obstacle large.
- Si aucun obstacle n'est détecté, le robot exécute simplement `avancer()` en continu.

7.3 Code final — contrôle Bluetooth (RemoteXY)

Ce code intègre l'interface RemoteXY (joystick) en conservant la sécurité du capteur à ultrasons : le robot n'avance pas sur commande du joystick si un obstacle est détecté trop proche.

```
#define REMOTEXY_MODE__SOFTSERIAL
#include <SoftwareSerial.h>
#define REMOTEXY_SERIAL_RX 2
#define REMOTEXY_SERIAL_TX 6
#define REMOTEXY_SERIAL_SPEED 9600
#include <RemoteXY.h>

#pragma pack(push, 1)
uint8_t const PROGMEM RemoteXY_CONF_PROGMEM[] =
{ 255,2,0,0,0,22,0,19,0,0,0,0,31,1,106,200,1,1,1,0,
  5,23,114,60,60,0,2,26,31 };

struct {
  int8_t joystick_01_x;
  int8_t joystick_01_y;
  uint8_t connect_flag;
} RemoteXY;
#pragma pack(pop)

// ===== Pins moteurs =====
#define IN1 4
#define IN2 5
#define IN3 7
#define IN4 8

// ===== Pins capteurs =====
#define TRIG 12
#define ECHO 13
#define IR_PIN A0

int seuilDistance = 15;

void avancer() {
  digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
```

```

    digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW);
}
void reculer() {
    digitalWrite(IN1, LOW); digitalWrite(IN2, HIGH);
    digitalWrite(IN3, LOW); digitalWrite(IN4, HIGH);
}
void tournerGauche() {
    digitalWrite(IN1, LOW); digitalWrite(IN2, HIGH);
    digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW);
}
void tournerDroite() {
    digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
    digitalWrite(IN3, LOW); digitalWrite(IN4, HIGH);
}
void arreter() {
    digitalWrite(IN1, LOW); digitalWrite(IN2, LOW);
    digitalWrite(IN3, LOW); digitalWrite(IN4, LOW);
}

long getDistance() {
    digitalWrite(TRIG, LOW); delayMicroseconds(2);
    digitalWrite(TRIG, HIGH); delayMicroseconds(10);
    digitalWrite(TRIG, LOW);
    long duration = pulseIn(ECHO, HIGH, 30000);
    if (duration == 0) return 999;
    return duration * 0.034 / 2;
}

void setup() {
    RemoteXY_Init();
    pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT);
    pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT);
    pinMode(TRIG, OUTPUT); pinMode(ECHO, INPUT);
    pinMode(IR_PIN, INPUT);
}

void loop() {
    RemoteXYEngine.handler();
    int x = RemoteXY.joystick_01_x;
    int y = RemoteXY.joystick_01_y;

    long distance = getDistance();
    bool obstacleProche = (distance < seuilDistance);

    if (y > 30 && !obstacleProche) { avancer(); }
    else if (y < -30) { reculer(); }
    else if (x > 30) { tournerDroite(); }
    else if (x < -30) { tournerGauche(); }
    else { arreter(); }
}

```

Explication des parties principales :

- `#define REMOTEXY_MODE__SOFTSERIAL` et `#include <SoftwareSerial.h>` : indiquent à la bibliothèque RemoteXY d'utiliser une liaison série logicielle (SoftwareSerial) plutôt que le port série matériel D0/D1, qui doit rester libre pour la programmation USB.
- `REMOTEXY_SERIAL_RX 2 / REMOTEXY_SERIAL_TX 6` : définissent les broches utilisées pour communiquer avec le module HC-05, conformément au câblage réalisé (TX du HC-05 vers D2, RX du HC-05 vers D6 via le diviseur de tension).

- Le bloc RemoteXY_CONF_PROGMEM[] est généré automatiquement par l'éditeur en ligne RemoteXY : il encode la disposition graphique de l'interface (ici, un joystick) sous une forme compacte que la bibliothèque sait interpréter pour reconstruire l'écran sur le smartphone.
- La structure RemoteXY { joystick_01_x; joystick_01_y; connect_flag; } définit les variables que le programme peut lire en continu : la position horizontale et verticale du joystick (de -100 à 100), et un indicateur de connexion. Ces variables sont automatiquement mises à jour par la bibliothèque dès qu'un mouvement est reçu depuis le téléphone.
- Les fonctions de mouvement (avancer, reculer, tournerGauche, tournerDroite, arreter) et la fonction getDistance() sont strictement identiques à celles du code d'évitement d'obstacles (section 7.2), réutilisées telles quelles.
- RemoteXY_Init() (dans setup()) initialise la communication Bluetooth et l'interface graphique.
- RemoteXYEngine.handler() (en tout début de loop()) doit être appelé à chaque passage de boucle : c'est lui qui traite les échanges Bluetooth en arrière-plan et tient à jour les variables joystick_01_x et joystick_01_y.
- La suite de la boucle traduit la position du joystick en commande de mouvement : une poussée vers le haut ($y > 30$) déclenche avancer(), mais uniquement si obstacleProche est faux — cette condition de sécurité empêche l'utilisateur de foncer volontairement dans un obstacle détecté par le capteur à ultrasons, même en pilotage manuel. Une poussée vers le bas ($y < -30$) déclenche reculer() sans condition (s'éloigner d'un danger reste toujours autorisé). Les mouvements latéraux du joystick ($x > 30$ ou $x < -30$) déclenchent les rotations. La zone centrale du joystick (entre -30 et 30 sur les deux axes) correspond à un arrêt.
- Les seuils de ± 30 créent une « zone morte » autour du centre du joystick, pour éviter que de petits tremblements de la main ne déclenchent des mouvements involontaires.

8. Difficultés rencontrées et solutions apportées

Problème rencontré	Solution apportée
Confusion sur le rôle des pins ENA / ENB après retrait des jumpers	Identification de la pin de signal réelle (vs. la pin reliée au 5V) par test progressif
Les 4 roues ne tournaient pas dans le même sens	Inversion physique des fils + et – sur les sorties OUT du L298N concernées
Le robot s'arrêtait dès le débranchement du câble USB	Câblage correct de l'alimentation batterie vers la broche Vin de l'Arduino (en plus du L298N)
Sensibilité du capteur infrarouge aux variations de lumière ambiante	Passage d'une logique « OU » à une logique de confirmation : le capteur à ultrasons reste prioritaire dans la décision
Incompatibilité de tension entre l'Arduino (5V) et le HC-05 (3,3V)	Ajout d'un diviseur de tension (2 résistances 1 k Ω en série) sur la ligne RX du module Bluetooth
Limitation de courant du module L298N avec 2 moteurs en parallèle par canal	Choix du module L298N standard (avec dissipateur) plutôt que la version mini ou le L293D, plus limités en courant

9. Conclusion

Ce projet a permis de transformer un kit pédagogique conçu pour un contrôleur propriétaire (ATX2) en un robot mobile entièrement piloté par un Arduino UNO, plus accessible et reprogrammable librement. Le résultat final est un robot fonctionnel, capable de se déplacer de manière autonome en évitant les obstacles grâce à un capteur à ultrasons complété par un capteur infrarouge, et pilotable à distance via Bluetooth depuis un smartphone grâce à l'application RemoteXY.

Ce travail illustre une démarche complète de prototypage électronique : choix et dimensionnement des composants, câblage, programmation, tests itératifs et résolution de problèmes concrets (inversions de polarité, gestion de l'alimentation, compatibilité de tensions logiques).