

UNIVERSITÉ MOULAY ISMAÏL — MEKNÈS ÉCOLE NORMALE SUPÉRIEURE

Master Spécialisé : Systèmes Intelligentes pour l'Éducation (SIE)

Module : Robotique Éducative et Applications

RAPPORT DE PROJET Conception et Réalisation d'un Robot Éducatif "EvoBot"

Réalisé par :

MOUSSAID Hicham
EL HANINE Amine

Encadré par :

Pr. A. REGRAGUI

Année Universitaire 2025-2026

Remerciements

Nous tenons à exprimer notre sincère gratitude à toutes les personnes ayant contribué de près ou de loin à la réalisation de ce projet.

Nos remerciements les plus chaleureux vont à notre professeur encadrant, **Monsieur Ahmed REGRAGUI**, pour la qualité de son enseignement, sa rigueur scientifique et sa disponibilité constante. Ses conseils avisés en conception mécanique et en électronique embarquée ont été déterminants dans la réussite de ce prototype.

Nous remercions également le corps professoral du Master SIE de l'École Normale Supérieure de Meknès pour la formation d'excellence dispensée tout au long de l'année.

Une pensée particulière pour le personnel technique du laboratoire et du FabLab universitaire, qui nous a facilité l'accès aux imprimantes 3D et aux outils de prototypage.

Enfin, nous remercions nos camarades de promotion pour l'esprit d'entraide et d'émulation qui a régné durant ces semaines de projet.

Table des matières

1	Introduction Générale	5
1.1	Contexte Pédagogique et Enjeux	5
1.2	Problématique	5
1.3	Objectifs du Projet	5
1.4	Ressources Numériques et Open-Source	5
2	Étude et Conception Mécanique	6
2.1	Outil de Conception : FreeCAD	6
2.2	Architecture du Robot	6
2.3	Dimensions Techniques du Châssis	7
2.3.1	1. Le Châssis Inférieur (Main Chassis)	7
2.3.2	2. Le Battery Bridge (Pont de Batterie)	7
2.3.3	3. Le Capot Supérieur (Top Lid)	8
2.4	Fabrication : Impression 3D	8
2.5	Assemblage et Post-traitement	8
3	Étude et Réalisation Électronique	8
3.1	Synoptique Global du Système	8
3.2	Le Cerveau : Arduino Uno R3	9
3.3	L'Interface de Puissance : Pont-H L298N	9
3.4	Motorisation et Mobilité	9
3.4.1	Moteurs à Réducteur (TT Motor)	9
3.4.2	Roues et Adhérence	10
3.5	Alimentation : Bloc 18650 Série	10
3.6	Les Capteurs (La Perception)	10
3.6.1	Capteur de Distance HC-SR04	10
3.6.2	Capteurs Infrarouges (Suivi de Ligne)	10
4	Développement Logiciel	12
4.1	Environnement de Développement : VS Code	12
4.2	Architecture Décisionnelle : Machine à États (FSM)	12
4.3	Architecture Fonctionnelle et Algorithmes	13
4.4	Détail de l'Algorithmme d'Évitement	13
4.5	Code Source Complet (Extrait Significatif)	14
5	Résultats, Limitations et Perspectives	15
5.1	Tests et Validation	15
5.2	Limitations et Problèmes Rencontrés	15
5.3	Perspectives d'Amélioration	15
6	Conclusion Générale	15
A	Annexe : Code Source Complet	16

B Annexe : Guide de Déploiement (VS Code)	20
B.1 Configuration de l'Environnement	20
B.2 Compilation et Téléchargement	20

Table des figures

1	Rendu 3D du robot EvoBot montrant l'assemblage final	6
2	Synoptique des flux d'information et de puissance	9
3	Schéma de câblage complet réalisé sous Fritzing	11
4	Architecture interne du robot éducatif	12
5	Logique d'Évitement d'Obstacles	13
6	Logique du Suiveur de Ligne	13

Liste des tableaux

1	Spécifications Géométriques de l'EvoBot	7
2	Paramètres d'Impression (Anycubic Vyper - PLA)	8

1 Introduction Générale

1.1 Contexte Pédagogique et Enjeux

La robotique éducative n'est pas seulement une discipline technique, c'est une méthode d'apprentissage transdisciplinaire. Elle incarne l'approche "STEM" (Science, Technology, Engineering, Mathematics) en permettant aux étudiants de matérialiser des concepts abstraits. Le projet **EvoBot** s'inscrit dans cette dynamique. Il vise à reproduire l'expérience d'apprentissage offerte par des robots commerciaux coûteux comme le *Thymio II*, mais en adoptant une philosophie "Maker" : **faire soi-même (DIY)**, comprendre et modifier.

1.2 Problématique

Les kits robotiques présents sur le marché souffrent souvent de deux défauts majeurs :

1. **Le Coût** : Un robot Thymio coûte environ 200€, ce qui limite l'équipement massif des salles de classe au Maroc.
2. **L'Opacité** : Ces robots sont souvent des "boîtes noires" impossibles à ouvrir ou à réparer sans outils spécialisés.

Notre défi est donc de concevoir une alternative "Open-Hardware" pour un budget inférieur à 500 MAD, tout en garantissant une robustesse suffisante pour une manipulation étudiante.

1.3 Objectifs du Projet

Le projet se décompose en objectifs techniques précis :

- **Conception CAO** : Modéliser un châssis original sous FreeCAD.
- **Fabrication Additive** : Maîtriser l'impression 3D pour produire les pièces.
- **Électronique** : Interfacer capteurs et actionneurs avec un microcontrôleur Arduino Uno.
- **Programmation** : Développer une intelligence artificielle séquentielle (Machine à États).

1.4 Ressources Numériques et Open-Source

Dans une volonté de partage et de transparence, l'intégralité des ressources du projet (modèles CAO, schémas électriques et code source) est disponible en ligne sur les dépôts GitHub officiels du projet :

Dépôts GitHub du Projet

- **El Hannine Amine** :
<https://github.com/L7A9/evobot>
- **Moussaïd Hicham** :
<https://github.com/moussaïdhicham/EvoBot-Educational-Robot>

2 Étude et Conception Mécanique

La mécanique est la fondation du robot. Elle doit protéger l'électronique et assurer la mobilité.

2.1 Outil de Conception : FreeCAD

Nous avons choisi **FreeCAD**, un logiciel de modélisation 3D paramétrique Open-Source. Contrairement à des outils comme Blender (orienté artistique), FreeCAD permet de définir des contraintes géométriques précises (distances, tangences). Cela nous a permis de concevoir un assemblage complexe où chaque trou de vis tombe exactement en face du composant réel.

2.2 Architecture du Robot

Le robot EvoBot est constitué de trois éléments imprimés distincts :

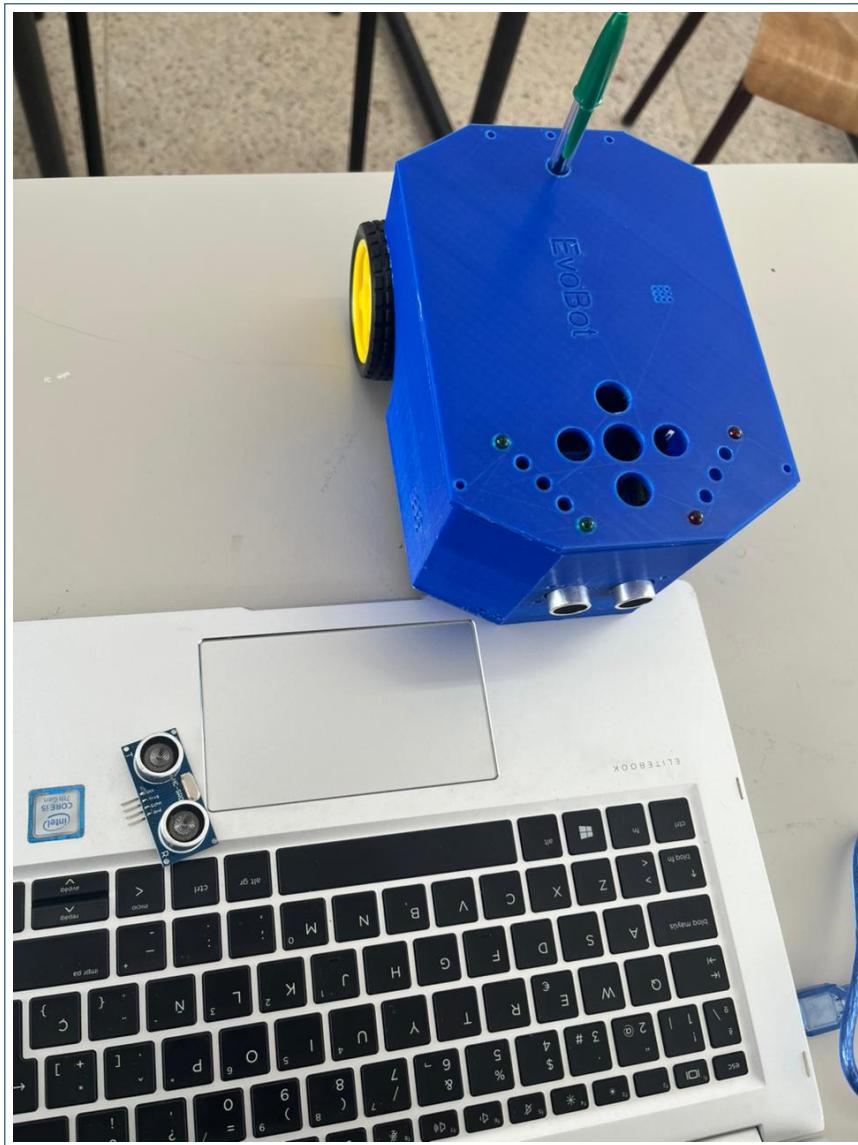


FIGURE 1 – Rendu 3D du robot EvoBot montrant l'assemblage final

2.3 Dimensions Techniques du Châssis

Le design de l'EvoBot a été optimisé pour la stabilité et la compacité. Voici les dimensions nominales extraites de notre modèle CAO :

TABLE 1 – Spécifications Géométriques de l'EvoBot

Composant	Dimension	Détail / Fonction
Longueur Hors-tout	160.0 mm	Encombrement longitudinal total.
Largeur Hors-tout	130.0 mm	Encombrement transversal total.
Hauteur Totale	80.0 mm	Hauteur châssis + capot monté.
Épaisseur Parois	3.0 mm	Équilibre entre poids et rigidité.
Rayon de Congé	15.0 mm	Angles arrondis pour la robustesse.
Garde au Sol (Roues)	10.0 mm	Dégagement suffisant pour les pistes.
Porte-Stylo	9.0 mm	Diamètre standard pour feutre scolaire.
Épaisseur Capot	3.5 mm	Renforcement de la partie supérieure.

2.3.1 1. Le Châssis Inférieur (Main Chassis)

C'est la pièce maîtresse. Imprimée en PLA noir, elle assure la rigidité structurelle.

- **Supports Moteurs** : Deux logements latéraux maintiennent les moteurs DC TT (jaunes) par friction ("Press-fit") et renforts vissés.
- **Fixation Arduino** : Des entretoises intégrées surélèvent la carte Arduino Uno pour éviter les courts-circuits avec le sol.
- **Capteurs Ligne** : Trois fentes sous le châssis permettent de positionner les modules infrarouges TCRT5000 à 3mm du sol, distance optimale pour la détection.

2.3.2 2. Le Battery Bridge (Pont de Batterie)

Face à la contrainte d'espace, nous avons innové avec le "Battery Bridge".

Focus : Le Battery Bridge

Cette pièce rouge vif forme un pont au-dessus de l'électronique centrale. Elle a une double fonction :

- **Support Énergétique** : Elle porte le boîtier de piles 18650 x 2, un composant lourd placé au centre de gravité pour une meilleure stabilité.
- **Gestion des Câbles** : L'espace sous le pont sert de "goulotte" pour cacher les dizaines de fils de connexion, rendant le montage final propre et sécurisé.

2.3.3 3. Le Capot Supérieur (Top Lid)

Le capot ferme le robot et porte l'interface utilisateur. Bien que non câblés dans cette version (limitations de l'Arduino Uno, voir Ch.5), nous avons prévu les emplacements pour :

- 5 Boutons poussoirs (Croix directionnelle + Centre).
- Un anneau de LEDs pour le feedback visuel.
- Un **Porte-stylo central** (Pen Hole) de 9mm de diamètre, permettant au robot de tracer sa trajectoire sur une feuille.

2.4 Fabrication : Impression 3D

Les pièces ont été produites par un procédé FDM (Fused Deposition Modeling) sur une imprimante **Anycubic Vyper**, reconnue pour sa précision et son niveling automatique performant.

TABLE 2 – Paramètres d’Impression (Anycubic Vyper - PLA)

Paramètre	Valeur	Justification
Matériau	PLA (Polylactic Acid)	Facile à imprimer, rigide, biodégradable.
Température Buse	205°C	Température optimale pour le PLA standard.
Vitesse d’impression	50 mm/s	Compromis entre état de surface et rapidité.
Remplissage (Infill)	20% (Grid)	Résistance structurelle optimisée.
Hauteur de Couche	0.2 mm	Détails précis pour les trous de vis.
Supports	Oui	Indispensables pour les porte-à-faux.

2.5 Assemblage et Post-traitement

Après l'impression, un travail de post-traitement a été nécessaire :

- **Ébavurage** : Retrait des supports sacrificiels au niveau des arches de roues et des passages de capteurs.
- **Taraudage léger** : Nettoyage des trous de vis M3 pour faciliter l'insertion des vis sans forcer sur le plastique.
- **Câblage interne** : Passage des fils dans la "goulotte" centrale sous le Battery Bridge avant le montage final du capot.

3 Étude et Réalisation Électronique

Cette section détaille les choix technologiques pour l'intelligence et la puissance.

3.1 Synoptique Global du Système

L'architecture électronique de l'EvoBot repose sur un échange continu entre perception et action. Le schéma fonctionnel ci-dessous illustre la circulation des signaux :

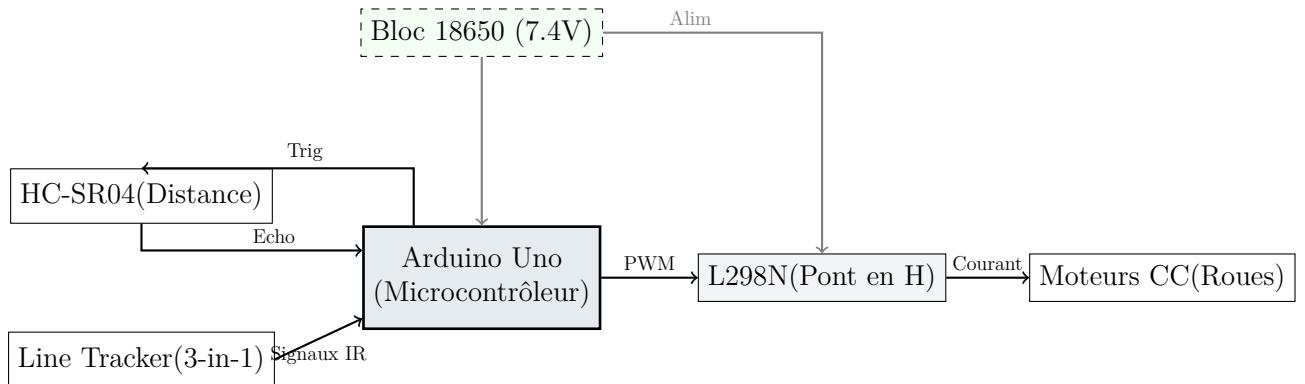


FIGURE 2 – Synoptique des flux d'information et de puissance

3.2 Le Cerveau : Arduino Uno R3

L'Arduino Uno est basé sur le microcontrôleur ATmega328P.

- **Fréquence** : 16 MHz.
- **Mémoire Flash** : 32 KO.
- **Tension Logique** : 5V.

C'est le standard éducatif mondial. Cependant, avec ses 14 E/S numériques, nous avons rapidement atteint ses limites (2 moteurs x 3 pins + Trig/Echo + 3 IR = 11 pins occupées sur 14).

3.3 L'Interface de Puissance : Pont-H L298N

Les moteurs consomment jusqu'à 2A en pic, ce qui détruirait l'Arduino (max 40mA). Le module L298N agit comme un amplificateur de courant. Il utilise deux Ponts en H : un assemblage de 4 transistors qui permet d'inverser la polarité aux bornes du moteur, et donc son sens de rotation.

3.4 Motorisation et Mobilité

Le déplacement de l'EvoBot repose sur deux moteurs à courant continu (CC) associés à des roues à haute adhérence.

3.4.1 Moteurs à Réducteur (TT Motor)

Nous avons choisi des moteurs **TT Jwn** avec réducteur plastique, réputés pour leur rapport qualité-prix en robotique mobile.

- **Rapport de réduction** : 1 :48 (offre un excellent compromis entre vitesse et couple).
- **Tension de fonctionnement** : 3V – 6V (optimisé pour nos 7,4V après chute de tension dans le L298N).
- **Couple maximal** : Environ 5,5 kg.cm à 6V, permettant de franchir de petits obstacles.
- **Consommation** : 70mA à vide, avec un pic à 250mA en charge bloquée.

3.4.2 Roues et Adhérence

Le robot est équipé de deux roues motrices de **65mm de diamètre** et **27mm de largeur**. La gomme en caoutchouc assure une traction optimale sur des surfaces lisses (carrelage, parquet) ou sur les pistes de suivi de ligne en papier.

3.5 Alimentation : Bloc 18650 Série

Pour garantir une autonomie suffisante et une tension stable de **7,4V**, nous avons opté pour un boîtier porte-piles spécifique :

- **Configuration** : 2 batteries Li-ion 18650 montées en série.
- **Sécurité** : Couvercle coulissant verrouillable par vis pour protéger les cellules des chocs et de l'environnement.
- **Pratичité** : Interrupteur ON/OFF intégré permettant de couper l'alimentation du robot sans débrancher les câbles.
- **Connectique** : Fils pré-soudés de 15cm facilitant le raccordement au driver L298N.

3.6 Les Capteurs (La Perception)

3.6.1 Capteur de Distance HC-SR04

Ce capteur fonctionne comme un sonar de chauve-souris.

1. Il émet une impulsion ultrason (40 kHz) via la broche *Trig*.
2. Il écoute l'écho via la broche *Echo*.
3. La distance est calculée par la formule : $d = \frac{v_{son} \times t_{echo}}{2}$ (avec $v_{son} \approx 340m/s$).

3.6.2 Capteurs Infrarouges (Suivi de Ligne)

Plutôt que des modules séparés, nous utilisons un **Infrared Line Tracker Sensor Module 3-in-1**. Ce module compact intègre trois paires d'émetteurs/récepteurs IR sur une seule carte, ce qui simplifie grandement l'alignement mécanique sous le châssis.

- Sur surface **BLANCHE** : Le signal infrarouge est réfléchi → Sortie LOW (0).
- Sur surface **NOIRE** : Le signal est absorbé → Sortie HIGH (1).
- **Réglage** : Un potentiomètre permet d'ajuster le seuil de détection selon l'éclairage ambiant.

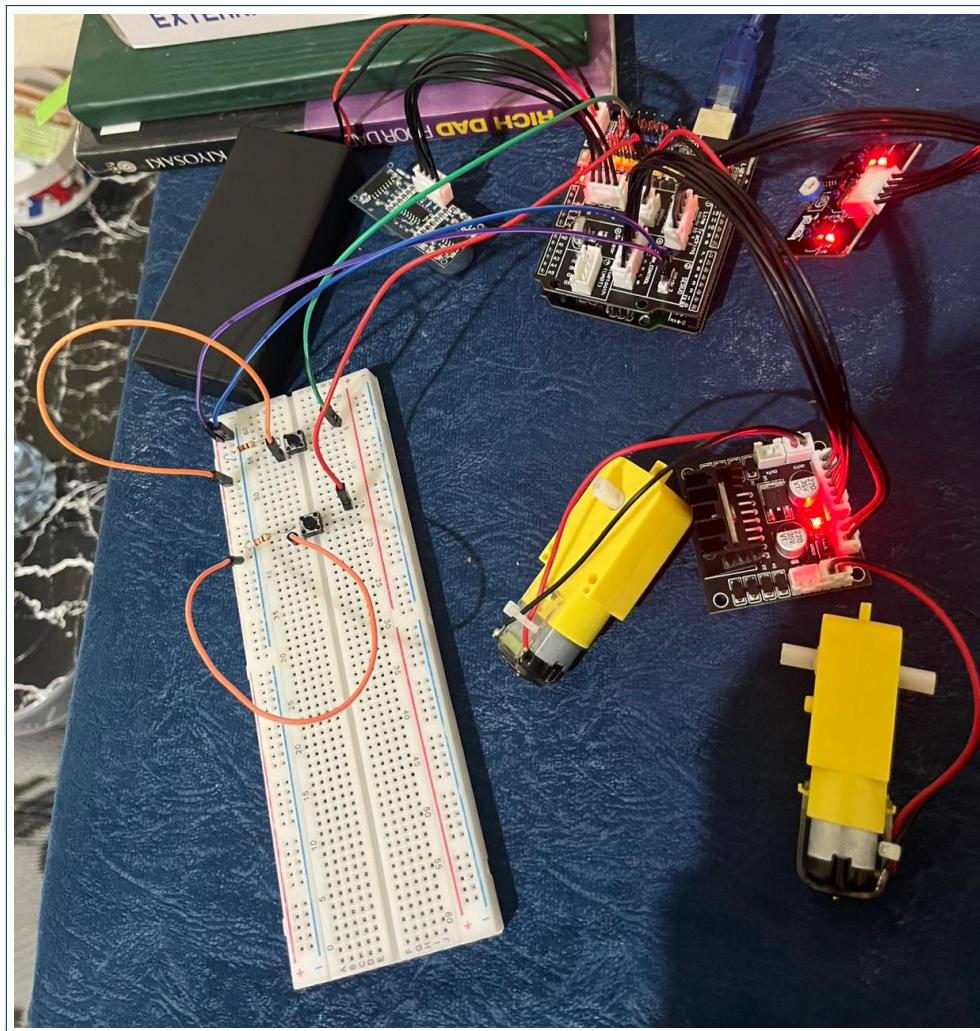


FIGURE 3 – Schéma de câblage complet réalisé sous Fritzing

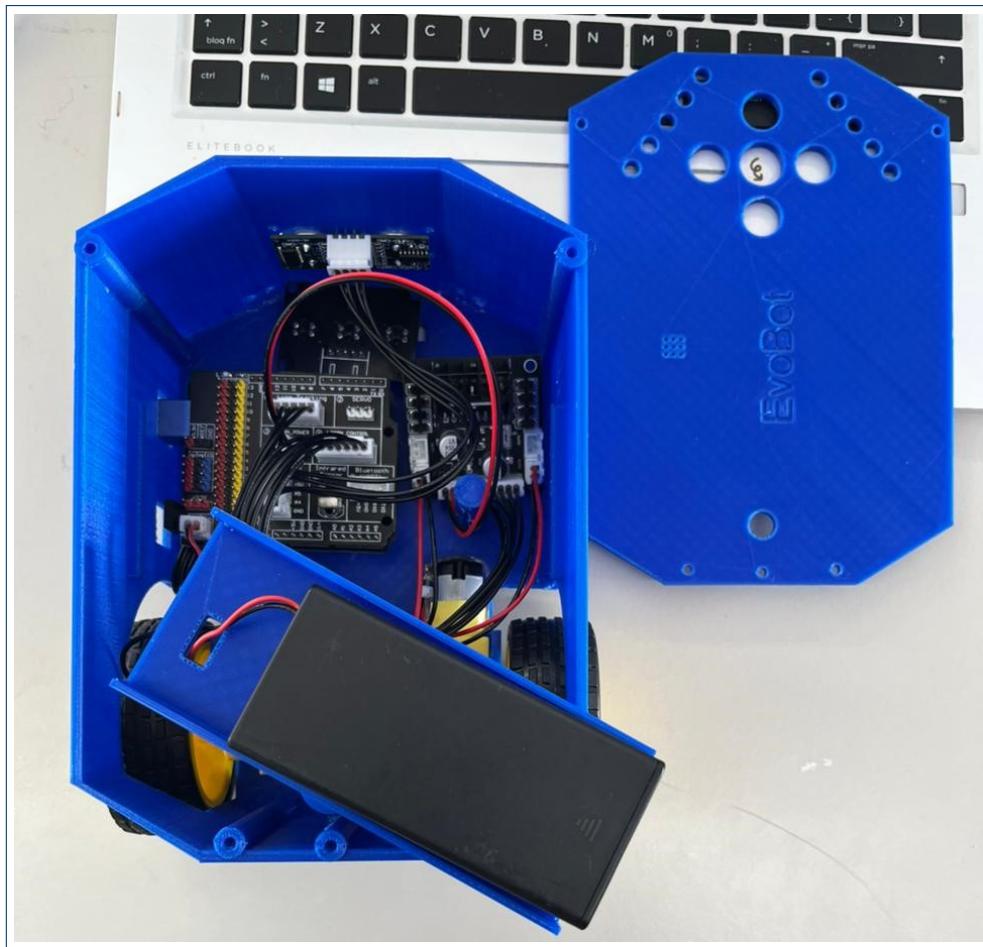


FIGURE 4 – Architecture interne du robot éducatif

4 Développement Logiciel

L'âme du robot réside dans son code. Nous avons opté pour une approche professionnelle.

4.1 Environnement de Développement : VS Code

Plutôt que l'IDE Arduino basique, nous avons utilisé **Visual Studio Code** avec l'extension **PlatformIO**.

- **Avantages** : Autocomplétion intelligente, gestion des bibliothèques plus propre, compilation plus rapide.
- **Structure C++** : Le code est organisé en fonctions claires (`move()`, `sense()`, `decide()`).

4.2 Architecture Décisionnelle : Machine à États (FSM)

Pour éviter le "Spaghetti Code", le robot est conçu comme une Machine à États Finis.

- **État 0 : IDLE**. Moteurs coupés. Attente appui bouton.
- **État 1 : LINE_FOLLOWING**. Boucle de régulation ultra-rapide.
- **État 2 : OBSTACLE_AVOIDANCE**. Navigation avec comportement réflexe.

4.3 Architecture Fonctionnelle et Algorithmes

L'EvoBot utilise deux logiques distinctes pour ses comportements principaux. Les diagrammes suivants illustrent le processus de décision en temps réel.

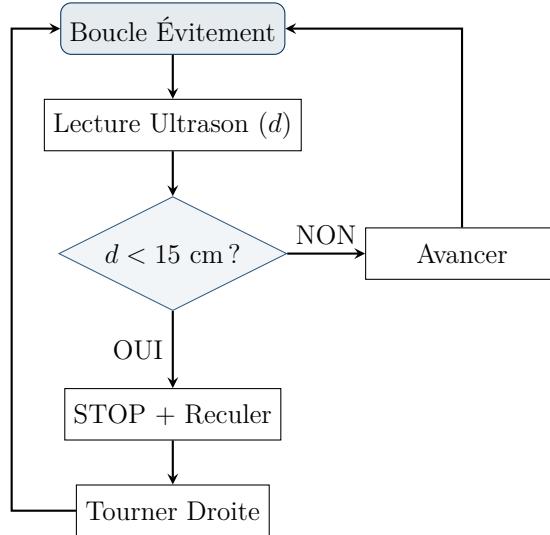


FIGURE 5 – Logique d’Évitement d’Obstacles

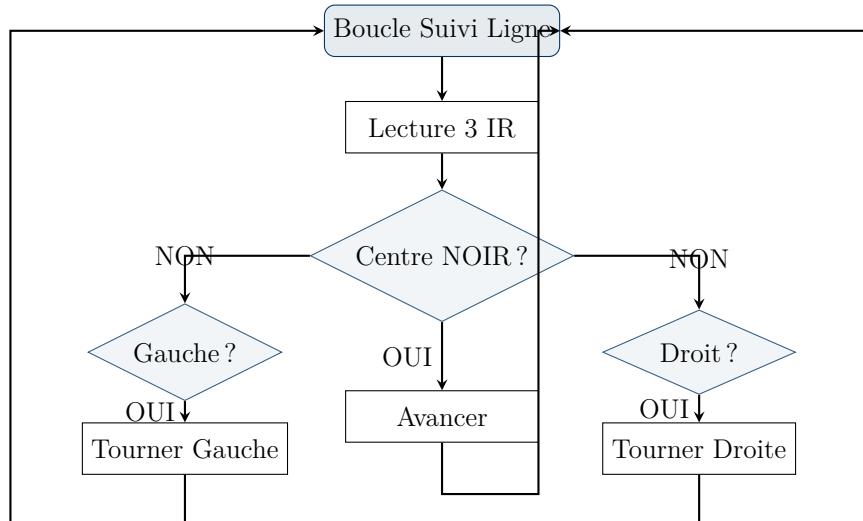


FIGURE 6 – Logique du Suiveur de Ligne

4.4 Détail de l’Algorithme d’Évitement

Voici le pseudo-code de la fonction de sécurité :

Algorithme : Evitement

Tant que (Mode == EVITEMENT) : 1. Lire Distance Ultrason d . 2. SI $d < 15$ cm (Obstacle!) : a. **STOP** (100ms) pour stabiliser. b. **REULER** (500ms) pour dégager la zone. c. **TOURNER DROITE** (300ms) pour chercher une issue. 3. SINON : a. **AVANCER** vitesse croisière. Fin Tant Que.

4.5 Code Source Complet (Extrait Significatif)

```

1 #include <Arduino.h>
2
3 // --- DEFINITION DES PINS ---
4 #define TRIG_PIN A5
5 #define ECHO_PIN A1
6 #define ENA 6 // Moteur Gauche PWM
7 #define IN1 11
8 #define IN2 9
9 // ... (autres definitions)
10
11 // --- VARIABLES GLOBALES ---
12 enum Mode { IDLE, LINE_FOLLOW, AVOIDANCE };
13 Mode currentMode = IDLE;
14
15 void setup() {
16     Serial.begin(9600); // Debugging
17     pinMode(TRIG_PIN, OUTPUT);
18     pinMode(ECHO_PIN, INPUT);
19     // Config moteurs en OUTPUT...
20 }
21
22 void loop() {
23     // 1. LECTURE BOUTONS
24     if (digitalRead(BTN_FOLLOW) == LOW) currentMode = LINE_FOLLOW;
25     if (digitalRead(BTN_AVOID) == LOW) currentMode = AVOIDANCE;
26
27     // 2. COMPORTEMENT SELON ETAT
28     switch(currentMode) {
29         case LINE_FOLLOW:
30             doLineFollow();
31             break;
32         case AVOIDANCE:
33             doAvoidance();
34             break;
35         default:
36             stopMotors();
37             break;
38     }
39 }
```

Listing 1 – Implémentation C++ principale

5 Résultats, Limitations et Perspectives

5.1 Tests et Validation

Nous avons soumis l'EvoBot à une série de tests fonctionnels, principalement par **détection palmaire** (test avec la main devant les capteurs). Cette phase de test manuelle était nécessaire pour isoler et valider la logique algorithmique avant le déploiement sur piste réelle.

1. **Test de Réactivité (Main)** : Les capteurs infrarouges réagissent immédiatement au passage de la main, déclenchant les mouvements moteurs correspondants.
2. **Validation Sonar** : Le capteur ultrason détecte la main et déclenche l'arrêt d'urgence et le recul, bien que des instabilités de mesure aient été notées (voir section 5.2).
3. **Autonomie Électrique** : Avec la batterie Li-ion, le système reste stable et alimenté durant les phases de tests prolongées.

5.2 Limitations et Problèmes Rencontrés

Deux obstacles majeurs ont été identifiés lors des phases de prototypage :

- **Instabilité du Capteur Ultrason (HC-SR04)** : Nous avons observé que le capteur ultrason renvoie parfois des valeurs erratiques. Cela semble être dû au bruit électromagnétique généré par les moteurs DC TT qui perturbe le signal d'écho. Un filtrage par moyenne glissante dans le code ou l'ajout de condensateurs de découplage sur les moteurs pourrait stabiliser ces mesures.
- **Connectique Arduino Uno** : Comme mentionné au Chapitre 2, le nombre limité de pins I/O empêche l'activation simultanée de tous les composants prévus dans le design (LEDs, boutons supplémentaires).

5.3 Perspectives d'Amélioration

Pour la prochaine version, nous recommandons deux évolutions majeures :

1. **Migration vers Arduino Mega 2560** : Cette carte possède 54 pins I/O. Elle permettrait de câbler l'intégralité du capot (LEDs, Buzzer, Stylo) sans compromis.
2. **Ajout d'un Module Bluetooth (HC-05)** : Pour permettre un pilotage ou une reprogrammation à distance via Smartphone.

6 Conclusion Générale

Ce projet de semestre a été une expérience d'ingénierie complète. Partis d'une feuille blanche, nous avons dû imaginer une solution mécanique (FreeCAD), choisir les composants électroniques adéquats, et les faire "vivre" grâce au code.

L'EvoBot est aujourd'hui une plateforme opérationnelle. Bien qu'il reste perfectible (notamment au niveau de la connectique Arduino Uno), il remplit son contrat initial : être un robot éducatif ouvert, réparable, et pédagogique.

A Annexe : Code Source Complet

Ci-dessous l'intégralité du code C++ téléversé sur le microcontrôleur Arduino Uno. Ce code gère la machine à états, le pilotage des moteurs via PWM et l'acquisition des données capteurs.

```

1 #include <Arduino.h>
2
3 // LINE SENSOR PINS
4 #define LEFT_SENSOR 2
5 #define MIDDLE_SENSOR 4
6 #define RIGHT_SENSOR 10
7
8 // ULTRASONIC SENSOR PINS
9 #define TRIG_PIN A5      // Changed to A5
10 #define ECHO_PIN A1     // Changed to A1
11
12 // BUTTON PINS
13 #define BUTTON_1_PIN 3  // Line following mode
14 #define BUTTON_2_PIN A0 // Obstacle avoidance mode
15
16 // L298N MOTOR PINS (PWM)
17 // Left motor (Motor A)
18 #define ENA 6          // PWM
19 #define IN1 11
20 #define IN2 9
21
22 // Right motor (Motor B)
23 #define ENB 5          // PWM
24 #define IN3 8
25 #define IN4 7
26
27 // SPEED SETTINGS
28 #define MOTOR_SPEED 150
29 #define TURN_SPEED 100
30 #define BACKUP_SPEED 100
31 #define AVOID_SPEED 130
32
33 // ULTRASONIC SETTINGS
34 #define OBSTACLE_DISTANCE 15 // cm - distance to detect obstacle
35 #define BACKUP_DISTANCE 8   // cm - distance to backup
36 #define BACKUP_TIME 500    // ms - time to backup
37 #define TURN_TIME 300     // ms - time to turn during avoidance
38
39 // OPERATION MODES
40 enum Mode {
41     MODE_IDLE,
42     MODE_LINE_FOLLOWING,
43     MODE_OBSTACLE_AVOIDANCE
44 };
45
46 Mode currentMode = MODE_IDLE;
47
48 // FUNCTION DECLARATIONS
49 void forward();
50 void backward();
51 void turnLeft();
52 void turnRight();
53 void stopMotors();
54 void setMotorSpeed(int leftSpeed, int rightSpeed);
55 void checkLineSensors();
56 void checkButtons();
57 float getDistance();
58 void obstacleAvoidance();
59 void backupAndTurn();
60
61 void setup() {
62     Serial.begin(9600);
63
64     // Line sensors
65     pinMode(LEFT_SENSOR, INPUT);

```

```

66  pinMode(MIDDLE_SENSOR, INPUT);
67  pinMode(RIGHT_SENSOR, INPUT);
68
69 // Ultrasonic sensor
70 pinMode(TRIG_PIN, OUTPUT);
71 pinMode(ECHO_PIN, INPUT);
72
73 // Buttons
74 pinMode(BUTTON_1_PIN, INPUT_PULLUP);
75 pinMode(BUTTON_2_PIN, INPUT_PULLUP);
76
77 // Motor pins
78 pinMode(ENA, OUTPUT);
79 pinMode(IN1, OUTPUT);
80 pinMode(IN2, OUTPUT);
81 pinMode(ENB, OUTPUT);
82 pinMode(IN3, OUTPUT);
83 pinMode(IN4, OUTPUT);
84
85 stopMotors();
86 Serial.println("System started. Press Button 1 for Line Following, Button 2 for
87 Obstacle Avoidance");
88
89 void loop() {
90     checkButtons();
91
92     switch(currentMode) {
93         case MODE_LINE_FOLLOWING:
94             checkLineSensors();
95             break;
96
97         case MODE_OBSTACLE_AVOIDANCE:
98             obstacleAvoidance();
99             break;
100
101        case MODE_IDLE:
102        default:
103            stopMotors();
104            break;
105    }
106
107    delay(50); // small delay for stability
108 }
109
110 //==== BUTTON CHECKING =====
111 void checkButtons() {
112     static bool lastButton1 = HIGH;
113     static bool lastButton2 = HIGH;
114
115     bool currentButton1 = digitalRead(BUTTON_1_PIN);
116     bool currentButton2 = digitalRead(BUTTON_2_PIN);
117
118     // Button 1 pressed (active LOW due to INPUT_PULLUP)
119     if (lastButton1 == HIGH && currentButton1 == LOW) {
120         currentMode = MODE_LINE_FOLLOWING;
121         Serial.println("Mode: Line Following");
122         delay(300); // Debounce delay
123     }
124
125     // Button 2 pressed (active LOW due to INPUT_PULLUP)
126     if (lastButton2 == HIGH && currentButton2 == LOW) {
127         currentMode = MODE_OBSTACLE_AVOIDANCE;
128         Serial.println("Mode: Obstacle Avoidance");
129         delay(300); // Debounce delay
130     }
131
132     lastButton1 = currentButton1;
133     lastButton2 = currentButton2;
134 }
135
136 //==== ULTRASONIC SENSOR FUNCTIONS =====
137 float getDistance() {

```

```

138 // Clear trigger pin
139 digitalWrite(TRIG_PIN, LOW);
140 delayMicroseconds(2);
141
142 // Send 10 s pulse to trigger
143 digitalWrite(TRIG_PIN, HIGH);
144 delayMicroseconds(10);
145 digitalWrite(TRIG_PIN, LOW);
146
147 // Read echo pulse duration
148 long duration = pulseIn(ECHO_PIN, HIGH);
149
150 // Calculate distance (cm)
151 float distance = duration * 0.034 / 2;
152
153 // Return distance or 0 if out of range
154 return (distance > 0 && distance < 400) ? distance : 0;
155 }
156
157 void backupAndTurn() {
158 Serial.println("Obstacle detected! Backing up and turning...");
159
160 // Backup
161 backward();
162 delay(BACKUP_TIME);
163
164 // Turn right
165 turnRight();
166 delay(TURN_TIME);
167
168 stopMotors();
169 delay(200);
170 }
171
172 void obstacleAvoidance() {
173 float distance = getDistance();
174
175 Serial.print("Distance: ");
176 Serial.print(distance);
177 Serial.println(" cm");
178
179 if (distance > 0 && distance <= OBSTACLE_DISTANCE) {
180 // Obstacle detected
181 backupAndTurn();
182
183 // Check again after turning
184 delay(100);
185 distance = getDistance();
186
187 // If still obstructed, keep turning
188 while (distance > 0 && distance <= OBSTACLE_DISTANCE) {
189 Serial.println("Still obstructed, turning more...");
190 turnRight();
191 delay(TURN_TIME);
192 stopMotors();
193 delay(100);
194 distance = getDistance();
195 }
196 } else {
197 // No obstacle, move forward
198 Serial.println("Clear path      Forward");
199 forward();
200 setMotorSpeed(AVOID_SPEED, AVOID_SPEED);
201 }
202 }
203
204 // ===== LINE SENSOR LOGIC =====
205 void checkLineSensors() {
206 int left   = digitalRead(LEFT_SENSOR);
207 int middle = digitalRead(MIDDLE_SENSOR);
208 int right  = digitalRead(RIGHT_SENSOR);
209
210 // Adjust HIGH / LOW if your sensors are inverted

```

```

211     if (middle == HIGH) {
212         Serial.println("Middle      Forward");
213         forward();
214     }
215     else if (left == HIGH) {
216         Serial.println("Left       Turn Left");
217         turnLeft();
218     }
219     else if (right == HIGH) {
220         Serial.println("Right      Turn Right");
221         turnRight();
222     }
223     else {
224         Serial.println("No line      Stop");
225         stopMotors();
226     }
227 }
228
229 //== MOTOR CONTROL ==
230 void setMotorSpeed(int leftSpeed, int rightSpeed) {
231     leftSpeed = constrain(leftSpeed, 0, 255);
232     rightSpeed = constrain(rightSpeed, 0, 255);
233
234     analogWrite(ENA, leftSpeed);
235     analogWrite(ENB, rightSpeed);
236 }
237
238 void forward() {
239     digitalWrite(IN1, HIGH);
240     digitalWrite(IN2, LOW);
241     digitalWrite(IN3, LOW);
242     digitalWrite(IN4, HIGH);
243
244     setMotorSpeed(BACKUP_SPEED, BACKUP_SPEED);
245 }
246
247 void backward() {
248     digitalWrite(IN1, LOW);
249     digitalWrite(IN2, HIGH);
250     digitalWrite(IN3, HIGH);
251     digitalWrite(IN4, LOW);
252
253     setMotorSpeed(BACKUP_SPEED, BACKUP_SPEED);
254 }
255
256 void turnLeft() {
257     digitalWrite(IN1, LOW);
258     digitalWrite(IN2, HIGH);
259     digitalWrite(IN3, LOW);
260     digitalWrite(IN4, HIGH);
261
262     setMotorSpeed(TURN_SPEED, MOTOR_SPEED);
263 }
264
265 void turnRight() {
266     digitalWrite(IN1, HIGH);
267     digitalWrite(IN2, LOW);
268     digitalWrite(IN3, HIGH);
269     digitalWrite(IN4, LOW);
270
271     setMotorSpeed(MOTOR_SPEED, TURN_SPEED);
272 }
273
274 void stopMotors() {
275     digitalWrite(IN1, LOW);
276     digitalWrite(IN2, LOW);
277     digitalWrite(IN3, LOW);
278     digitalWrite(IN4, LOW);
279
280     setMotorSpeed(0, 0);
281 }

```

Listing 2 – Firmware EvoBot - main.cpp

B Annexe : Guide de Déploiement (VS Code)

Pour exécuter le code de l'EvoBot et le téléverser sur la carte Arduino, nous recommandons l'utilisation de **Visual Studio Code** avec l'extension **PlatformIO**.

B.1 Configuration de l'Environnement

1. Télécharger et installer **VS Code**.
2. Aller dans le menu *Extensions* (Ctrl+Shift+X) et chercher "**PlatformIO IDE**". L'installer.
3. Ouvrir le dossier racine du projet via *File > Open Folder...*. PlatformIO reconnaîtra automatiquement le fichier `platformio.ini`.

B.2 Compilation et Téléchargement

- **Compiler** : Cliquer sur l'icône "Check" () en bas à gauche pour vérifier le code.
- **Téléverser** : Brancher l'Arduino en USB et cliquer sur l'icône "Flèche" (→) pour envoyer le code.
- **Moniteur Série** : Utiliser l'icône "Prise électrique" pour voir les messages de debug (Distance, Mode actif).