Présentation du Projet Rescue Junior :

(Marius FELICETTI - BUT1 GEII - et Sagesse FONTAREL - BUT2 GEII Parcours AII -)

Introduction :

Ce document détaille les étapes du développement d'un projet de robotique visant à concevoir un robot capable de se déplacer de manière autonome et de suivre une ligne noire. Il décrit les choix technologiques, la méthodologie, et le programme utilisé pour atteindre les objectifs.

---

1. Analyse Réflexive sur la Pertinence du Choix

Dans le cadre de ce projet, le choix des composants matériels et des approches méthodologiques joue un rôle crucial.

1.1 Composants Choisis

- **Carte Arduino Uno** : Simplicité d'utilisation et large communauté pour le support technique.
- **MotoDriver 2** : Contrôle robuste de deux moteurs, permettant une manipulation flexible.
- **Moteurs** : Inconnu.

1.2 Approche Méthodologique

La méthode adoptée repose sur des itérations successives :

1. Développement d'une version initiale pour valider les concepts.
2. Ajustements en fonction des performances observées.
3. Améliorations continues basées sur les retours et l'analyse.

Cette approche garantit un système final optimisé et robuste.

2. Programme pour le Robot

Le projet est divisé en deux grandes parties :

1. Faire avancer et tourner le robot.
2. Repérer une ligne noire grâce à un capteur.

2.1 Faire Avancer et Tourner le Robot

Responsable : Marius
Objectif : Permettre au robot de se déplacer en avant et de tourner.

Assemblage du Matériel

- **Composants Utilisés** :
    - Carte Arduino Uno
    - Driver Moteur MotoDriver 2
    - 2 moteurs

Étape 1 : Réalisation du câblage

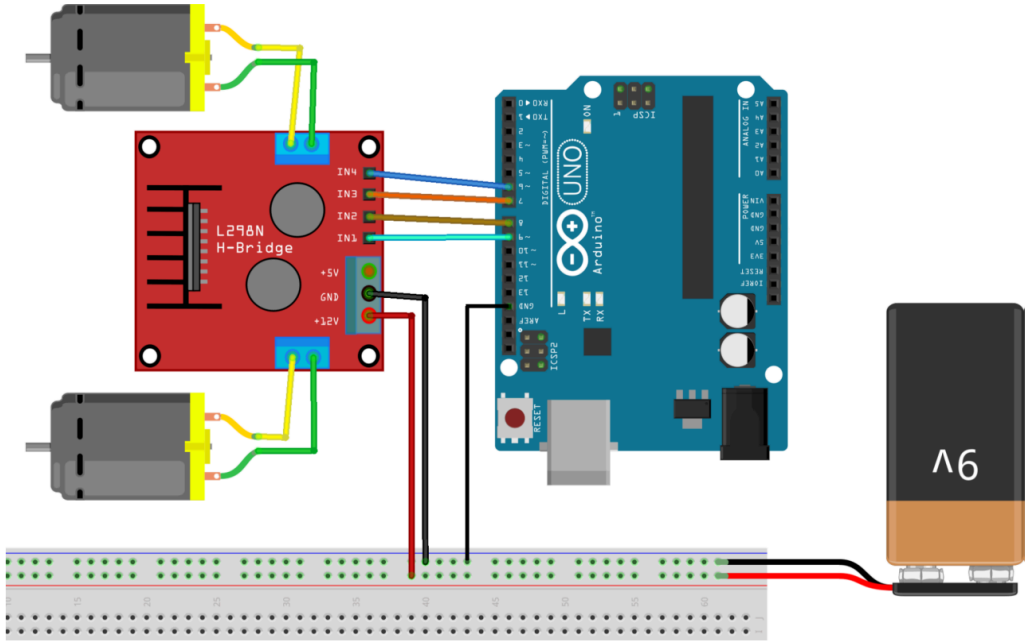
Pour réaliser le câblage, il faut :

1. Connecter la carte Arduino Uno avec le driver moteur en utilisant 4 câbles mâle-femelle.
2. Relier le driver avec les 2 moteurs avec 2 câbles mâle-mâle pour chaque moteur.
3. Fournir une alimentation via une pile 9V reliée à une plaque d'essai Arduino pour alimenter tous les composants du montage.

NB :

- Le MotoDriver 2 possède un régulateur 5V intégré.
- Le montage suit les instructions du manuel constructeur **MotoDriver 2**, qui contient également un exemple de code pour contrôler les moteurs.

Voici ci-dessous le montage pour contrôler nos 2 moteurs :



NB :

-> Le MotoDriver2 à un régulateur 5V intégré

Le montage vient du manuel constructeur suivant : MotoDriver2. Il contient également un exemple de code permettant de contrôler les moteurs.

Exemple d'un autre montage

Étape 2 : Programmation

Écrire un programme Arduino pour contrôler les roues, afin de permettre au robot d'avancer et de tourner.

- Le programme contient plusieurs fonctions, chacune liée à une action spécifique (vitesse de rotation des roues, sens de rotation des roues, etc.).

Résultats des Tests :

1. **Premier essai** :
    - Après avoir réalisé le câblage et téléversé le programme de test dans la carte Arduino, les LEDs d'alimentation du driver moteur et de la carte Arduino s'allument, mais les moteurs ne tournent pas.

    - Après changement de pile :
        - Le test sur un des deux moteurs fonctionne.
        - Le moteur tourne bien, mais consomme rapidement la pile. Le moteur ne tourne pas toujours.
2. **Deuxième essai : Alimentation à partir d'une prise murale**
    - En alimentant un moteur avec une prise de courant, le moteur tourne, mais la chenille empêche un mouvement fluide.
    - Lorsque la chenille est retirée, la roue tourne mieux mais pas parfaitement. Il faut le lancer à la main et il a peu de puissance. (valable pour les deux moteurs).

Idée :

- Utiliser une pile dédiée pour alimenter la carte Arduino, qui ne dispose actuellement d'aucune alimentation propre (hormis l'ordinateur).

3. **Test suivant :**
    - Le problème venait de l'alimentation insuffisante en courant via la prise murale.
    - Solution : remplacer cette alimentation par celle d'un ordinateur.
    - Résultat : les moteurs fonctionnent parfaitement et tournent avec beaucoup de puissance.

Partie Capteur (Sagesse) : Suivre une Ligne Noire

1. Objectif :

Créer un robot capable de suivre une ligne noire en détectant sa position avec un capteur de couleur et en ajustant sa trajectoire en conséquence.

---

2. Étapes Chronologiques du Projet

2.1 Compréhension et Configuration Initiale

- **Composants utilisés :**
    - **Capteur de couleur TCS34725 RGB**
    - **Microcontrôleur (Arduino Uno ou équivalent)**
- **Première étape : Configurer le matériel**
    - Connecter le capteur de couleur au microcontrôleur.
    - Vérifier la compatibilité logicielle (bibliothèques nécessaires).
    - Effectuer un premier test pour s'assurer que le capteur renvoie des données valides.

Problème rencontré : Bibliothèque Adafruit introuvable.

- **Cause possible :** La bibliothèque n'est pas installée ou est mal configurée.
- **Solution :**
    1. Rechercher et installer la bibliothèque Adafruit_TCS34725 via le gestionnaire de bibliothèques d'Arduino.
    2. Vérifier que le fichier .h de la bibliothèque est bien inclus dans le code avec #include <Adafruit_TCS34725.h>.

2.2 Lecture des Données du Capteur

1. Écrire un programme pour lire les données de lumière et détecter la ligne noire.
    - Mesurer les valeurs de lumière réfléchie par la surface.
    - Définir un seuil de luminosité correspondant à la ligne noire.
2. Tester en environnement réel avec des variations d'éclairage.

Problème rencontré : Difficulté à distinguer la ligne noire en cas d'éclairage non uniforme.

- **Cause possible :** La lumière ambiante perturbe les mesures.
- **Solution :**
    1. Réaliser un calibrage initial dans l'environnement d'utilisation.
    2. Implémenter une fonction de correction automatique qui ajuste les seuils selon les variations de lumière.

2.3 Contrôle des Mouvements

1. **Logique de base :**
    - Si le capteur détecte la ligne noire au centre, le robot avance tout droit.
    - Si le capteur détecte un écart à gauche ou à droite, le robot ajuste sa trajectoire :
        - Tourner à gauche : Ralentir la roue gauche et accélérer la roue droite.
        - Tourner à droite : Ralentir la roue droite et accélérer la roue gauche.
2. **Lissage des mouvements :**
    - Appliquer un facteur proportionnel (Kp) pour rendre les corrections progressives.
    - Utiliser la distance au centre de la ligne pour moduler les ajustements.

Problème rencontré : Incapacité à savoir si le robot doit tourner à gauche ou à droite.

- **Cause possible :** Le capteur unique ne peut pas distinguer les deux côtés.
- **Solution :**
    1. Ajouter deux capteurs latéraux en plus du capteur central.
    2. Implémenter une logique :
        - Si le capteur gauche détecte la ligne noire, tourner à gauche.
        - Si le capteur droit détecte la ligne noire, tourner à droite.

2.4 Tests et Validation

1. **Tester le capteur :**
    - Scotchs de différentes couleurs
    - Patchs colorés de taille variable.

---

3. Problèmes Supplémentaires et Résolutions

3.1 Perte de la ligne noire

- **Cause possible :** Les seuils de détection sont mal ajustés.
- **Solution :**
    - Implémenter une fonction de recherche automatique pour retrouver la ligne noire lorsque le robot s'en éloigne.

3.2 Oscillations autour de la ligne

- **Cause possible :** Ajustements trop rapides ou trop lents.
- **Solution :**
    - Ajuster le facteur proportionnel (Kp) et ajouter un terme dérivé (Kd) pour réduire les oscillations.

---

4. Analyse Réflexive

Ce que le projet nous a permis de comprendre :

1. **L'importance de la modularité :** Découper le projet en étapes claires a permis d'identifier et de résoudre les problèmes efficacement.
2. **Les défis de l'environnement réel :** Les variations d'éclairage et les imprécisions mécaniques nécessitent une adaptation constante des seuils et des paramètres.
3. **L'approche algorithmique :** La mise en œuvre d'un contrôle proportionnel a démontré l'importance d'utiliser des modèles mathématiques simples pour résoudre des problèmes pratiques.
4. **Le travail en équipe :** La répartition des tâches (Sagesse pour les capteurs, Marius pour le contrôle des moteurs) a facilité une meilleure spécialisation et efficacité.

---

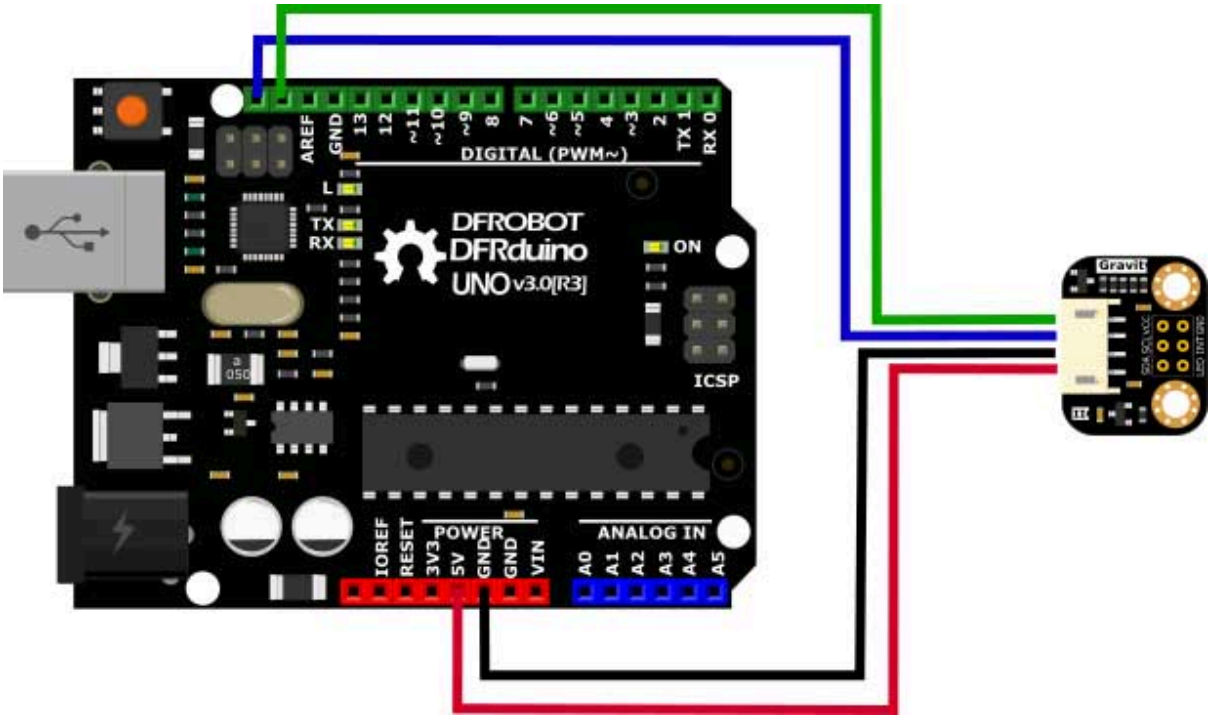
5. Perspectives d'Amélioration

- Ajouter des capteurs supplémentaires pour détecter les intersections ou obstacles.
- Intégrer un algorithme pour ajuster automatiquement les paramètres en fonction des conditions.
- Améliorer le système d'alimentation pour plus d'autonomie.

Conclusion :

Ce projet met en œuvre une combinaison de matériel et de programmation pour concevoir un robot autonome capable de naviguer efficacement. Grâce à une approche

méthodologique itérative, il est possible d'optimiser les performances et d'assurer le succès du projet.




```
//project 2 RGB sensor w/ IR
#include <Wire.h>
#include "Adafruit_TCS34725.h"
const int IR_PIN = 10; // Pin connected to the HW201 IR module OUT
const int LED_PIN = 9; // Pin connected to the onboard LED control pin of the TCS34725
// Create an instance of the TCS34725 sensor
Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS,
TCS34725_GAIN_1X);
bool objectDetected = false;
void setup() {
Serial.begin(9600);
pinMode(IR_PIN, INPUT);
pinMode(LED_PIN, OUTPUT);
digitalWrite(LED_PIN, LOW); // Turn off LED initially
if (tcs.begin()) {
Serial.println("Found TCS34725 sensor");
// Ensure the LED is off by default
tcs.setInterrupt(true);
} else {
Serial.println("No TCS34725 found ... check your connections");
while (1); // halt the program
}
}
void loop() {
```

```
// Read the IR sensor
int irValue = digitalRead(IR_PIN);
if (irValue == LOW) { // Assuming LOW means object detected
if (!objectDetected) {
objectDetected = true;
Serial.println("Object detected, taking color reading...");
delay(1000); //delay for 1sec before turn on LED
// Turn on the LED
digitalWrite(LED_PIN, HIGH);
delay(500); // Give some time for the LED to stabilize
uint16_t r, g, b, c;
tcs.getRawData(&r, &g, &b, &c);
// Normalize the values
float sum = c;
float red = r / sum;
float green = g / sum;
float blue = b / sum;
Serial.print("R: "); Serial.print(r); Serial.print(" ");
Serial.print("G: "); Serial.print(g); Serial.print(" ");
Serial.print("B: "); Serial.print(b); Serial.print(" ");
Serial.print("C: "); Serial.print(c); Serial.println(" ");
// Calculate the color temperature and lux
uint16_t colorTemp = tcs.calculateColorTemperature(r, g, b);
uint16_t lux = tcs.calculateLux(r, g, b);
Serial.print("Color Temp: "); Serial.print(colorTemp, DEC); Serial.print(" K - ");
Serial.print("Lux: "); Serial.println(lux, DEC);
// Determine the color
if (red > green && red > blue) {
Serial.println("Detected Color: RED");
} else if (green > red && green > blue) {
Serial.println("Detected Color: GREEN");
} else if (blue > red && blue > green) {
Serial.println("Detected Color: BLUE");
} else {
Serial.println("Detected Color: UNKNOWN");
}
// Turn off the LED
digitalWrite(LED_PIN, LOW);
delay(1000); // Wait for some time before the next reading
}
} else {
if (objectDetected) {
objectDetected = false; // Reset object detected flag
}
delay(100); // Small delay to prevent rapid loop execution
}
}
```

https://github.com/adafruit/Adafruit_TCS34725
https://downloads.arduino.cc/libraries/github.com/adafruit/Adafruit_TCS34725-1.4.4.zip

Voici une mise à jour du programme intégrant la logique pour avancer, tourner, et s'ajuster en fonction du décalage par rapport à la ligne blanche. Les actions des moteurs sont inspirées de votre dossier,

Ainsi pour le Semestre 4 nous aurons une base sur laquelle continuer le projet Rescue.

Code

```
#include <Wire.h>
#include "Adafruit_TCS34725.h"

// Initialisation du capteur TCS34725
Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_50MS, TCS34725_GAIN_4X);

// Déclaration des broches pour les moteurs
#define MD1 2
#define MD2 3
#define MG1 4
#define MG2 5

void setup() {
  Serial.begin(9600);

  pinMode(MD1, OUTPUT);
  pinMode(MD2, OUTPUT);
  pinMode(MG1, OUTPUT);
  pinMode(MG2, OUTPUT);

  if (tcs.begin()) {
    Serial.println("Capteur trouvé");
  } else {
    Serial.println("Capteur non détecté");
    while (1);
  }
}

void avancer() {
  digitalWrite(MD1, HIGH);
  digitalWrite(MD2, LOW);
  digitalWrite(MG1, HIGH);
  digitalWrite(MG2, LOW);
  Serial.println("Commande : Avancer");
```

```
}

void tournerDroite() {
  digitalWrite(MD1, HIGH);
  digitalWrite(MD2, HIGH);
  digitalWrite(MG1, HIGH);
  digitalWrite(MG2, LOW);
  Serial.println("Commande : Tourner à droite");
}

void pivoterDroite() {
  digitalWrite(MD1, LOW);
  digitalWrite(MD2, HIGH);
  digitalWrite(MG1, HIGH);
  digitalWrite(MG2, LOW);
  Serial.println("Commande : Pivoter à droite");
}

void tournerGauche() {
  digitalWrite(MD1, HIGH);
  digitalWrite(MD2, LOW);
  digitalWrite(MG1, HIGH);
  digitalWrite(MG2, HIGH);
  Serial.println("Commande : Tourner à gauche");
}

void pivoterGauche() {
  digitalWrite(MD1, HIGH);
  digitalWrite(MD2, LOW);
  digitalWrite(MG1, LOW);
  digitalWrite(MG2, HIGH);
  Serial.println("Commande : Pivoter à gauche");
}

void ajusterMoteurs(int offset) {
  if (offset > 50) { // Grand décalage à droite
    pivoterGauche();
  } else if (offset < -50) { // Grand décalage à gauche
    pivoterDroite();
  } else if (offset > 10) { // Léger décalage à droite
    tournerGauche();
  } else if (offset < -10) { // Léger décalage à gauche
    tournerDroite();
  } else { // Centré
    avancer();
  }
}
```

```
int calculerDecalage(uint16_t r, uint16_t g) {
  return (int)g - (int)r; // Vert - Rouge (positif = droite, négatif = gauche)
}

void loop() {
  uint16_t r, g, b, c;

  // Lire les données brutes du capteur
  tcs.getRawData(&r, &g, &b, &c);

  // Affichage des données lues
  Serial.print("Rouge: ");
  Serial.print(r);
  Serial.print(" Vert: ");
  Serial.print(g);
  Serial.print(" Bleu: ");
  Serial.print(b);
  Serial.print(" Luminosité claire: ");
  Serial.println(c);

  // Calcul du décalage
  int offset = calculerDecalage(r, g);

  // Afficher le décalage
  Serial.print("Décalage calculé : ");
  Serial.println(offset);

  // Ajuster les moteurs en fonction du décalage
  ajusterMoteurs(offset);

  delay(200); // Délai pour stabiliser
}
```

Fonctionnement conjecturer : (à vérifier au S4)

1. **Actions des moteurs** :
    - Les fonctions avancer(), tournerDroite(), etc., sont définies selon votre documentation pour contrôler un pont en H.
    - Les broches MD1, MD2, MG1, MG2 contrôlent les deux moteurs.
2. **Détection du décalage** :
    - La différence entre les valeurs de vert et rouge (g - r) est calculée.
    - Si l'offset est positif, le robot est à droite de la ligne blanche.
    - Si l'offset est négatif, le robot est à gauche.

3. **Ajustement** :
    - Grand décalage : pivoter pour revenir sur la ligne.
    - Petit décalage : ajuster légèrement la direction.
    - Centré : avancer tout droit.
4. **Calibration** :
    - Ajustez les seuils pour offset (e.g., ±10, ±50) en fonction des conditions réelles.
5. **Test et Validation** :
    - Vérifiez les couleurs détectées et la réponse des moteurs.
    - Assurez-vous que le robot corrige efficacement sa trajectoire.

Ce programme associe les actions moteur à la logique de correction pour maintenir le robot aligné sur la ligne blanche.