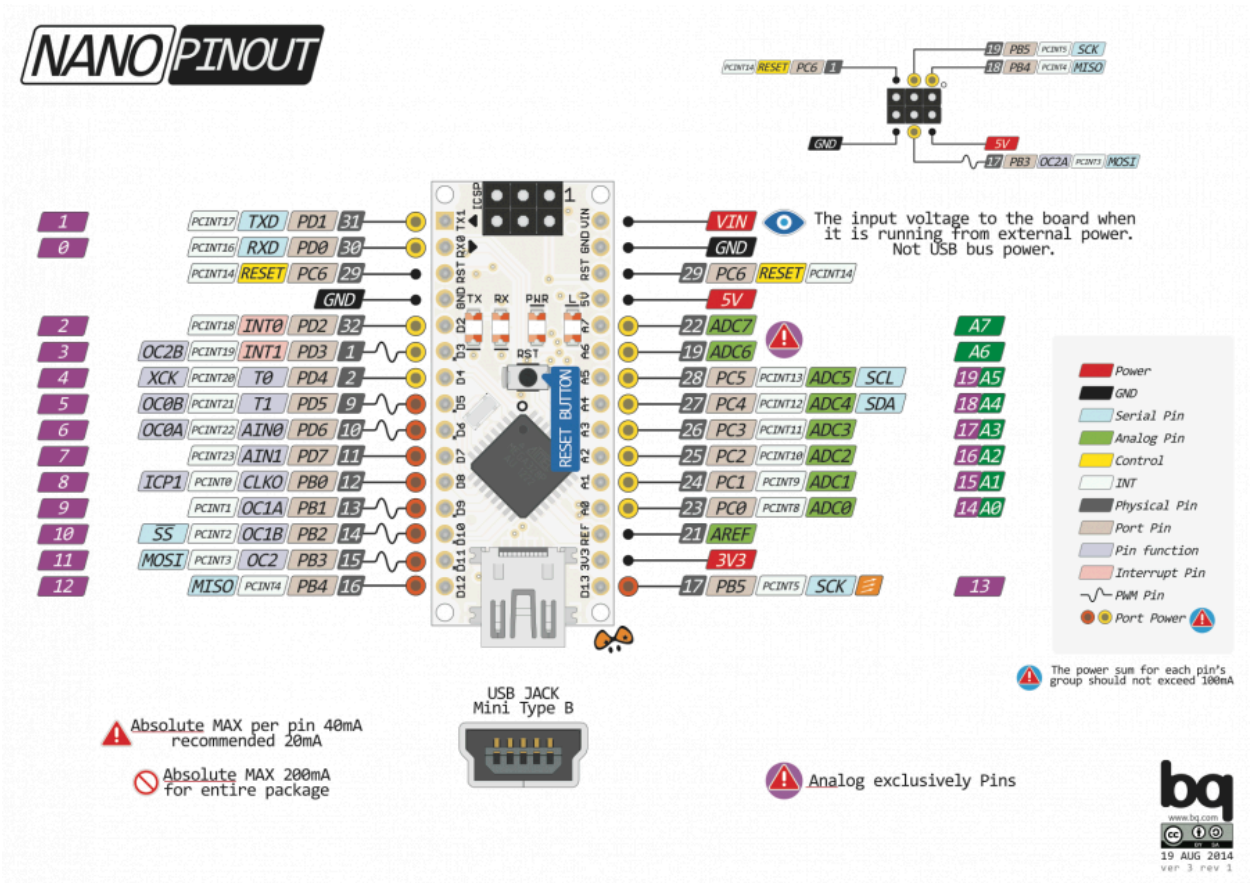
Générale



Quantité	Nom du Composant	Description / Notes
2	VNH7070	Moteur
2	VL53L1X	Lidar de distance
1	LED RGB	Barrette
1	Bouton Démarrage	-;
1	Batterie	Mesure de tension
1	MPU9250	Boussole
-	RX / TX	Indisponible sauf pour Débug
2 (Analogique)	CNY70	capteur reflectif
2 (TOR)	CNY70	capteur reflectif

Trois cartes à réaliser

Carte	Fonction	Étudiant
carte avec capteurs photoreflectifs	capteurs photoreflectifs	Arda
carte drivers moteur	drivers moteur	Daniel
carte shield arduino nano	shield arduino nano	Baptiste

Pinout Arduino Nano v3.xw

Pin	Fonction	Utilisation
D0/RX	Digital (Réception Série)	
D1/TX	Digital (Transmission Série)	
D2	Digital	IN_A_D
D3	Digital (PWM, Interrupt)	IN_B_D
D4	Digital	
D5	Digital (PWM)	
D6	Digital (PWM)	
D7	Digital	IN_A_G
D8	Digital	IN_B_G
D9	Digital (PWM)	MLI_D
D10	Digital (PWM, SPI SS)	MLI_G
D11	Digital (PWM, SPI MOSI)	Xshut_G
D12	Digital (SPI MISO)	Xshut_D
D13	Digital (SPI SCK)	BP_Démarrage
A0	Analogique/Digital	led_data
A1	Analogique/Digital	C_P_D

Pin	Fonction	Utilisation
A2	Analogique/Digital	C_P_G
A3	Analogique/Digital	Bandeau_led
A4	Analogique/Digital (I2C SDA)	I2C SDA
A5	Analogique/Digital (I2C SCL)	I2C SCL
A6	Analogique uniquement	C_M_G
A7	Analogique uniquement	C_M_D
VIN	Alimentation externe (7-12V)	
+5V	Sortie Alimentation 5V	
3V3	Sortie Alimentation 3.3V	
GND	Masse	
AREF	Référence Analogique Externe	
RESET	Réinitialisation	

1 : Daniel (Moteur)

N°	Symbole	Fonctions	ARDUINO | Indépendant	Entrées(Valeurs à Envoyer)/Sorties
4, 5, 12	**V_CC**	Broches d'alimentation de puissance du circuit.	Indépendant	**(Au 12V)**
1, 16	**GND_A**	Source du transistor bas (low-side) du côté A.	Indépendant	**(Au GND)**
8, 9	**GND_B**	Source du transistor bas (low-side) du côté B.	Indépendant	**(Au GND)**
3	**IN_A**	Entrée de commande logique pour le sens horaire (Clockwise).	ARDUINO	**Entrées 0/1 (Stop/Sens Horaire)**
6	**IN_B**	Entrée de commande logique pour le sens anti-horaire (Counter clockwise).	ARDUINO	**Entrées 0/1 (Stop/Sens Anti-Horaire)**
2, 15	**OUT_A**	Sortie A du pont. Elle correspond à la source du transistor haut (high-side) A et au drain du transistor bas (low-side) A.	**MOTEUR**	**Sorties [Moteur +]**
7, 10	**OUT_B**	Sortie B du pont. Elle correspond à la source du transistor haut (high-side) B et au drain du transistor bas (low-side) B.	**MOTEUR**	**Sorties [Moteur -]**
11	**PWM**	Entrée de commande en modulation de largeur d'impulsion (compatible CMOS, avec hystérésis). Elle permet de moduler le ratio d'état (Haut/Bas ou Bas/Haut) des transistors bas durant leur phase d'activation afin de contrôler la vitesse du moteur.	ARDUINO	**Entrées [MLI Nano] (Timer 1)**
13	**CS**	Sortie de diagnostic analogique multiplexée (Current Sense). Elle délivre un courant proportionnel au courant du moteur en fonction de la branche sélectionnée.	Indépendant	**Sorties [Inutilisé] (Au GND)**
14	**SEL_0**	Entrée de sélection (active à l'état haut, compatible CMOS 3 V et 5 V). En combinaison avec IN_A et IN_B, elle sert à adresser l'information de diagnostic (CurrentSense) renvoyée au microcontrôleur.	Indépendant	**Sorties [Inutilisé] (Au GND)**

|| CODE DE TEST MOTEUR || :

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

//D7 A_G | D8 B_G
//D2 B_D | D3 A_D
//IN_A_G & IN_B_D pour Avancer

int main() {
 // broche en sortie
 DDRB |= (1 << PB0);
 DDRB |= (1 << PB1) | (1 << PB2);  //MLI MOTEUR G | D
 DDRD |= (1 << PD7);
 // timer1 en mode 14
 TCCR1B |= (1 << WGM13) | (1 << WGM12);
 TCCR1A |= (1 << WGM11) | (1 << COM1A1) | (1 << COM1B1);
 // valeur du prédiviseur
 // 8P et OCR1 = 8000
 TCCR1B |= (1 << CS11);  // 8 en Prédiviseur
 ICR1 |= 99;
 // générer le signal MLI sur la broche OC1A ou OC1B (bits COM1xx dans le
registre TCCR1A)
 // valeur du rapport cyclique
 OCR1A |= 49;  // OCR1A = 0.50*ICR1
 OCR1B |= 49;  // OCR1B = 0.50*ICR1
 sei();
 while (1) {
   //MOTEUR GAUCHE
   PORTD |= (1 << PD7);  //IN_A à 1
   OCR1A = 50;           //Moteur G à 50%
   _delay_ms(750);
   OCR1A = 25;  //Moteur G à 25%
   _delay_ms(750);
   PORTD &= ~(1 << PD7);  //IN_A à 0
   PORTB |= (1 << PB0);   //IN_B à 1
   OCR1A = 25;            //Moteur G à 25%
   _delay_ms(750);
   OCR1A = 50;  //Moteur G à 50%
```

```
    _delay_ms(750);
    PORTB &= ~(1 << PB0);  //IN_B à 0
    _delay_ms(1000);
    //MOTEUR DROITE
    PORTD |= (1 << PD3);  //IN_A à 1
    OCR1B = 50;           //Moteur D à 50%
    _delay_ms(750);
    OCR1B = 25;  //Moteur D à 25%
    _delay_ms(750);
    PORTD &= ~(1 << PD3);  //IN_A à 0
    PORTD |= (1 << PD2);   //IN_B à 1
    OCR1B = 25;            //Moteur D à 25%
    _delay_ms(750);
    OCR1B = 50;  //Moteur D à 50%
    _delay_ms(750);
    PORTD &= ~(1 << PD2);  //IN_B à 0
    _delay_ms(1000);

    //MOTEUR FREINS
    PORTD |= (1 << PD7);  //IN_A_G à 1
    OCR1A = 100;          //Moteur D à 100%
    PORTD |= (1 << PD2);  //IN_B_D à 1
    OCR1B = 100;          //Moteur G à 100%
    _delay_ms(750);
    PORTD |= (1 << PD7);  //IN_A_G à 1
    PORTD |= (1 << PD3);  //IN_A_D à 1
    PORTB |= (1 << PB0);  //IN_B_G à 1
    PORTD |= (1 << PD2);  //IN_B_D à 1
    _delay_ms(750);
    PORTD &= ~(1 << PD7);  //IN_A_G à 0
    PORTD &= ~(1 << PD3);  //IN_A_D à 0
    PORTB &= ~(1 << PB0);  //IN_B_G à 0
    PORTD &= ~(1 << PD2);  //IN_B_D à 0
  }
}
```

Code Suivie Ligne Jeudi Midi :

```
#include <Arduino.h>
#include <avr/io.h>
#include <util/delay.h>
```

```
#include <avr/interrupt.h>
#include <math.h>      // pour log()
#include <Wire.h>      // Communication I2C
#include <VL53L1X.h>  // Bibliothèque du capteur de distance
#include <PololuLedStrip.h>

PololuLedStrip<A0> rubanLed;
#define nbLeds 8
rgb_color tableauCouleurs[nbLeds]; // tableau des couleurs des LEDs

// ---------- Broches XSHUT ----------
const int xshut1 = 11;  // Capteur 1
const int xshut2 = 12;  // Capteur 2

VL53L1X lidar1;  // Premier capteur
VL53L1X lidar2;  // Deuxième capteur

// --- Configuration des Seuils de Ligne ---
const int SEUIL_NOIR = 850;
const int SEUIL_temps_Charge_Noir = 27500;

// --- Variables pour fonctions moteurs ---
int ligne_g = 0;
int ligne_d = 0;

// Machine d'état
enum state {
etapeBatterie, // État initial : vérification batterie & attente bouton
etapeButton,   // État intermédiaire (si besoin)
etapeLigne     // Suivi de ligne actif
};
state etapeActive = etapeBatterie;

// --- Fonctions de contrôle des moteurs ---
void moteurGauche(int vitesse, bool avance, bool recule) {
 if (avance) PORTD |= (1 << PD7);
 else PORTD &= ~(1 << PD7);

 if (recule) PORTB |= (1 << PB0);
 else PORTB &= ~(1 << PB0);
```

```
 OCR1A = vitesse;
}

void moteurDroite(int vitesse, bool avance, bool recule) {
 if (avance) PORTD |= (1 << PD2);
 else PORTD &= ~(1 << PD2);

 if (recule) PORTD |= (1 << PD3);
 else PORTD &= ~(1 << PD3);

 OCR1B = vitesse;
}

void ADC_init() {
// Référence AVcc (5V)
ADMUX = (1 << REFS0);
// Activation ADC + prescaler 128
ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
}

uint16_t lireADC(uint8_t canal) {
ADMUX = (ADMUX & 0xF0) | (canal & 0x0F);
ADCSRA |= (1 << ADSC);
while (ADCSRA & (1 << ADSC));
return ADC;
}

unsigned long mesurerTempsDecharge(uint8_t pinBit) {
 DDRC |= (1 << pinBit);
 PORTC |= (1 << pinBit);
 delayMicroseconds(50);

 DDRC &= ~(1 << pinBit);
 unsigned long debut = micros();

 while ((PINC & (1 << pinBit)) && (micros() - debut < 30000)) {}
 return micros() - debut;
}
```

```
// --- Programme Principal ---
int main() {
 // 1. DIRECTION DES BROCHES (Sorties moteurs)
 DDRB |= (1 << PB0) | (1 << PB1) | (1 << PB2);
 DDRD |= (1 << PD7) | (1 << PD3) | (1 << PD2);

 // 2. CONFIGURATION STRICTE DU TIMER 1 À 20 kHz
 TCCR1A = 0;
 TCCR1B = 0;
 TCCR1A |= (1 << WGM11) | (1 << COM1A1) | (1 << COM1B1);
 TCCR1B |= (1 << WGM13) | (1 << WGM12);
 ICR1 = 799;
 TCCR1B |= (1 << CS10);

 OCR1A = 0;
 OCR1B = 0;
 PORTD &= ~((1 << PD7) | (1 << PD3) | (1 << PD2));
 PORTB &= ~(1 << PB0);

 // 3. ARDUINO INITIALISATION
 init();
 ADC_init(); // Crucial pour que la mesure batterie fonctionne !
 DDRC &= ~((1 << PC1) | (1 << PC2));

 bool brake = false;

 // --- CONFIGURATION DU BOUTON PB5 (Après init() pour éviter
l'écrasement) ---
 DDRB |= (1 << PB5);
 PORTB |= (1 << PB5);

 // --- INITIALISATION LIDARS ---
 Wire.begin();
 Wire.setClock(400000);
  pinMode(xshut1, OUTPUT);
 pinMode(xshut2, OUTPUT);
 digitalWrite(xshut1, LOW);
 digitalWrite(xshut2, LOW);
 delay(100);
  digitalWrite(xshut1, HIGH);
```

```
 delay(100);
 lidar1.setTimeout(500);
 if (!lidar1.init()) {
   while (1);
 }
 lidar1.setAddress(0x2A);
 lidar1.setDistanceMode(VL53L1X::Long);
 lidar1.startContinuous(50);
  digitalWrite(xshut2, HIGH);
 delay(100);
 lidar2.setTimeout(500);
 if (!lidar2.init()) {
   while (1);
 }
 lidar2.setDistanceMode(VL53L1X::Long);
 lidar2.startContinuous(50);
  sei();

 TCCR0A = (1 << WGM01) | (1 << WGM00);
 TCCR0B = (1 << CS01) | (1 << CS00);
 TIMSK0 = (1 << TOIE0);

 int distance1 = 500; // Valeur par défaut (voie libre)

 // 4. BOUCLE PRINCIPALE
 while (1) {

   // Lecture du LiDAR non-bloquante
   if (lidar1.dataReady()) {
     distance1 = lidar1.read();
   }

   // Gestion globale du bouton pour arrêter le robot à tout moment
   // Si on appuie sur le bouton alors qu'on roule, on coupe tout et on
retourne à la batterie
   if (etapeActive == etapeLigne && !(PINB & (1 << PB5))) {
     moteurGauche(0, false, false);
     moteurDroite(0, false, false);
     etapeActive = etapeBatterie;
     _delay_ms(300); // Anti-rebond
```

```
  }

  switch (etapeActive) {

    case etapeBatterie:
      {
        moteurGauche(0, false, false);
        moteurDroite(0, false, false);

        float batt = lireADC(3) * (5.07 / 1023.0);

        if (batt < 3.0) {
          for (int i = 0; i < nbLeds; i++) tableauCouleurs[i] =
rgb_color(10, 0, 0);  // Rouge (Vide)
        } else if (batt < 3.5) { // Ajusté à 3.5V pour une vraie mesure
LiPo/LiFe
          for (int i = 0; i < nbLeds; i++) tableauCouleurs[i] =
rgb_color(10, 3, 0);  // Orange (Faible)
        } else {
          for (int i = 0; i < nbLeds; i++) tableauCouleurs[i] =
rgb_color(0, 10, 0);  // Vert (OK)
        }
        rubanLed.write(tableauCouleurs, nbLeds);

        // Action du BOUTON : Passe directement au suivi de ligne
        if (!(PINB & (1 << PB5))) {
          _delay_ms(300); // Anti-rebond
          etapeActive = etapeLigne; // On lance le robot !
        }
      }
      break;

    case etapeButton:
      // Espace réservé si tu as besoin d'une étape intermédiaire
      break;

    case etapeLigne:
      if (brake){
        moteurGauche(0, true, true);
        moteurDroite(0, true, true);
```

```
      }
      else {

        if (distance1 >= 250) { // Si champ libre à plus de 25 cm
        ligne_g = analogRead(A6);
        ligne_d = analogRead(A7);

        int v = 200; // Vitesse de base minimale

        // Logique de suivi de ligne
        if (ligne_g > 750 && ligne_d > 750) {
          moteurGauche((ligne_g / 2) - 150, true, false);
          moteurDroite((ligne_d / 2) - 150, true, false);
        } else {
          moteurGauche(((ligne_g / 6) + (v / 2)), true, false);
          moteurDroite(((ligne_d / 6) + (v / 2)), true, false);
        }
      }
      else { // Obstacle détecté à moins de 25 cm -> Freinage d'urgence
        moteurGauche(0, true, true);
        moteurDroite(0, true, true);
        brake = 1;
      }
      }
      break;
   }
 }
}
```

```
// Capteurs Marques
          tempsChargeA1 = mesurerTempsDecharge(PC1);
          tempsChargeA2 = mesurerTempsDecharge(PC2);
// --- LOGIQUE DES MARQUES ---
          if (tempsChargeA1 > SEUIL_temps_Charge_Noir && A1_Precedent <
SEUIL_temps_Charge_Noir) {
            Droite = Droite + 1;
          }
          if (tempsChargeA2 > SEUIL_temps_Charge_Noir && A2_Precedent <
SEUIL_temps_Charge_Noir) {
            Gauche = Gauche + 1;
          }

          // Sauvegarde des états pour le flanc montant au prochain tour
          A1_Precedent = tempsChargeA1;
          A2_Precedent = tempsChargeA2;

const int SEUIL_temps_Charge_Noir = 27500;  // tempsChargeNoir = 30000
```

CA :

```
// --- Variables pour fonctions moteurs ---
int ligne_g = 0;
int ligne_d = 0;
float Kp = 0.2;     // Gain proportionnel
int consigne_ligne = 0;    // Consigne pour le suivi de ligne (erreur
cible = 0 pour être centré)

if (ligne_g > 50 || ligne_d > 50 )
        {
          tempsChargeA1 = mesurerTempsDecharge(PC1);
          tempsChargeA2 = mesurerTempsDecharge(PC2);

          int v = 275;
          erreur = (ligne_g - ligne_d) - consigne_ligne;
          action = (Kp * erreur);

          int vitesseG = v + action;
          int vitesseD = v - action;
```

```
          moteurGauche(vitesseG, true, false);
          moteurDroite(vitesseD, true, false);
} else {
            moteurGauche(0, true, true);
            moteurDroite(0, true, true);
            etapeActive = brake;
     }
```

DROITE ET GAUCHE :

```
case tournergauchenoir:
       moteurGauche(400, false, true);
       moteurDroite(400, true, false);
       if (ligne_g < 100 || ligne_d < 100 )
       {
         etapeActive = tournergaucheblanc;
         _delay_ms(5);
       }
       break;

     case tournerdroitenoir:
         moteurGauche(400, true, false);
         moteurDroite(400, false, true);
         if (ligne_g < 100 || ligne_d < 100 )
         {
           etapeActive = tournerdroiteblanc;
           _delay_ms(5);
         }
       break;

     case tournergaucheblanc:
         moteurGauche(300, false, true);
         moteurDroite(300, true, false);
         if (ligne_g > 300 && ligne_d > 300 )
         {
           etapeActive = Ligne;
           _delay_ms(5);
         }
       break;
```

```
    case tournerdroiteblanc:
        moteurGauche(300, true, false);
        moteurDroite(300, false, true);
        if (ligne_g > 300 || ligne_d > 300 )
        {
          etapeActive = Ligne;
          _delay_ms(5);
        }
      break;
```

Logique Marques :

```
const uint8_t NOMBRE_INTERSECTION_G = 3;
const uint8_t NOMBRE_INTERSECTION_D = 0;
const int intersectionG[NOMBRE_INTERSECTION_G] = {5,8,10};
const int intersectionD[NOMBRE_INTERSECTION_D] = {};
```

```
tempsChargeA1 = mesurerTempsDecharge(PC1);
          tempsChargeA2 = mesurerTempsDecharge(PC2);
```

```
if (tempsChargeA1 > SEUIL_temps_Charge_Noir && A1_Precedent <
SEUIL_temps_Charge_Noir) {
            Droite = Droite + 1;
          }
          if (tempsChargeA2 > SEUIL_temps_Charge_Noir && A2_Precedent <
SEUIL_temps_Charge_Noir) {
            Gauche = Gauche + 1;
          }

          Serial.print("  Gauche = ");  Serial.println(Gauche);
          Serial.print(" | Droite = "); Serial.println(Droite);

          if (Gauche == intersectionG[x]){
            etapeActive = tournergauchenoir;
            x = x + 1;
          }
          if (NOMBRE_INTERSECTION_D > 0 && Droite == intersectionD[y]){
            etapeActive = tournergauchenoir;
            y = y + 1;
          }
```

```
            A1_Precedent = tempsChargeA1;
            A2_Precedent = tempsChargeA2;
```

2 : Baptiste (Principale)

1. ARCHITECTURE MATÉRIELLE

Description des Broches du Composant MPU-9250

Ce tableau détaille les fonctions et les spécifications techniques de chaque broche pour l'intégration dans le projet.

Étiquette	Fonction de la Broche	Remarques
SCL	Horloge série I2C / Horloge port série SPI	100 ou 400 kHz pour I2C. Jusqu'à 1 MHz pour SPI (20 MHz dans certains cas).
SDA	Données série I2C	Peut également être utilisé pour l'entrée de données série SPI (SDI).
VDD	Alimentation	+2,4V à +3,6V.
GND	Référence de masse	0V.
AUXDA	Données série maître I2C	Pour la connexion à des capteurs externes.
FSYNC	Entrée numérique de synchronisation	Connecter à la masse (GND) si inutilisé.
AUXCL	Horloge série maître I2C	Pour la connexion à des capteurs externes.
INT	Signal d'interruption	Sortie numérique d'interruption (totem pole ou drain ouvert).

Étiquette	Fonction de la Broche	Remarques
CS	Sélection de puce (Chip Select)	Utilisé uniquement en mode SPI.
AD0 / SDO	Sélection d'adresse / Sortie SPI	I2C (AD0) : Low = 0x68, High = 0x69. SPI (SDO) : Sortie de données série.
VDDIO	Alimentation des broches d'E/S	+1,71V jusqu'à VDD.

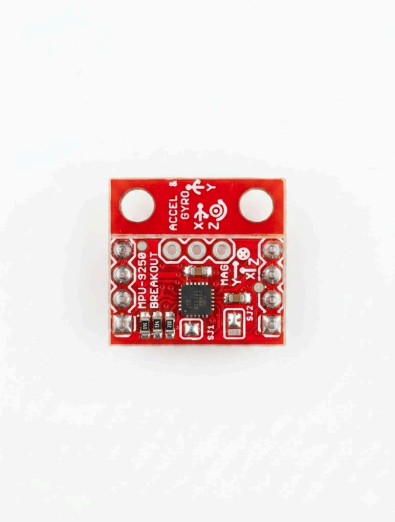
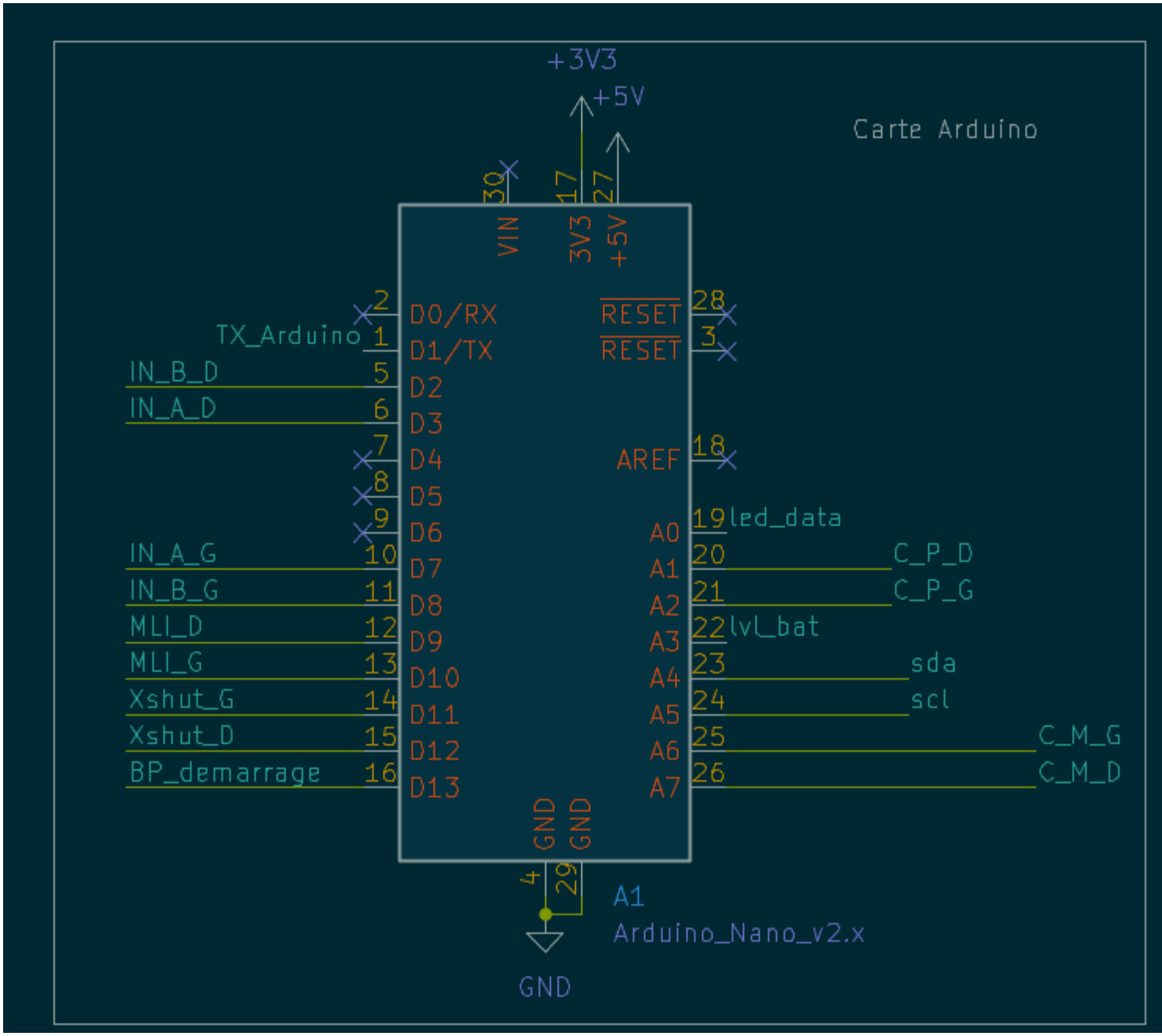


Schéma Kicad

- **Microcontrôleur (Carte Arduino Nano v2.x) :** C'est le cerveau de la carte. Il est responsable de la logique de contrôle et de la gestion des périphériques.

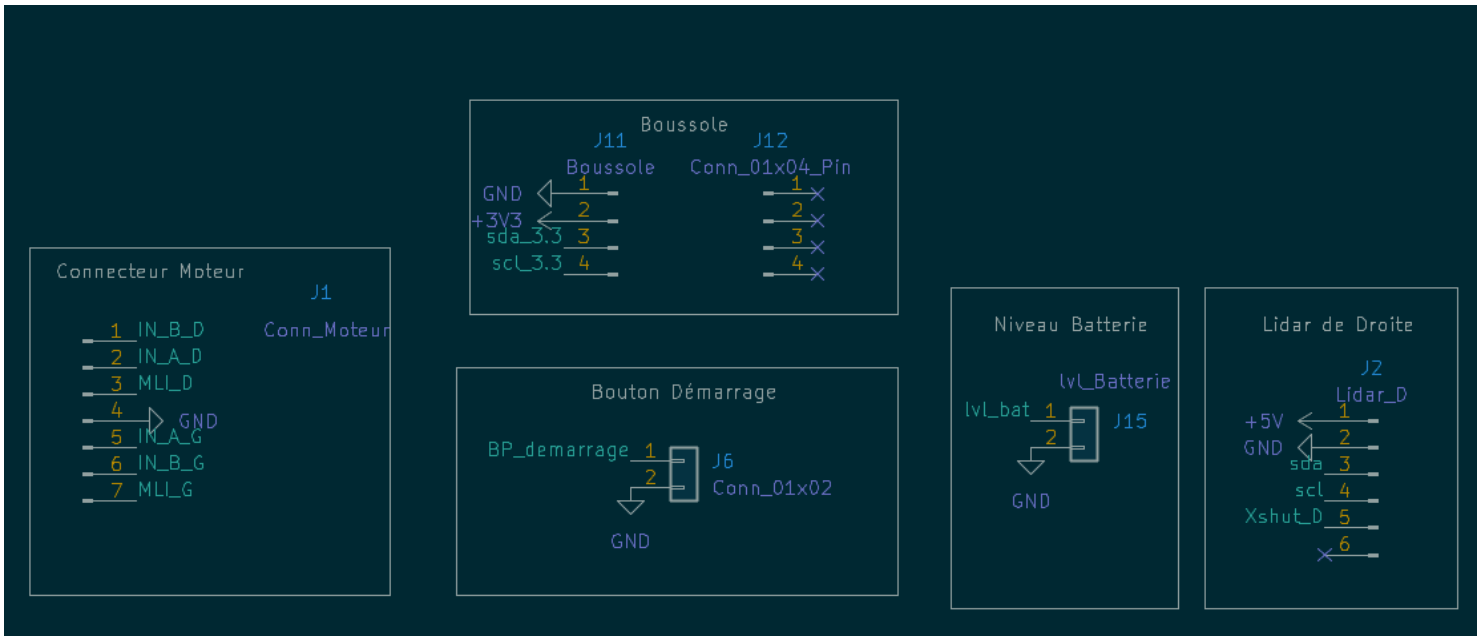


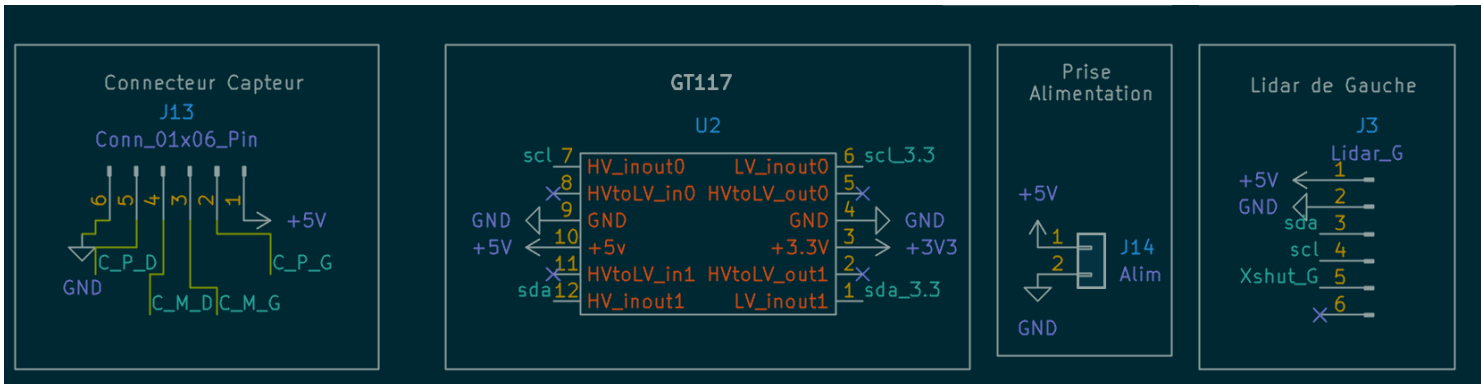
- **Connecteurs de Puissance et de Commande :**
  - **Moteurs (J13) :** Ce connecteur six broches (Conn_01x06_Pin) est dédié à la commande des moteurs. Les signaux incluent des entrées/sorties pour le contrôle des moteurs droit et gauche (IN_A_D, IN_B_D, IN_A_G, IN_B_G) et les signaux de modulation de largeur d'impulsion (MLI_D, MLI_G) pour la variation de vitesse.
  - **Périphériques I2C/SPI (Capteurs) :** Des connecteurs spécifiques sont prévus pour les capteurs. Les lignes *scl* et *sda* gèrent la communication

série, notamment pour la **Boussole** (J12) et les **Lidars Droit et Gauche** (J16 et J17). Le signal *Xshut* permet le contrôle d'activation/désactivation des Lidars.

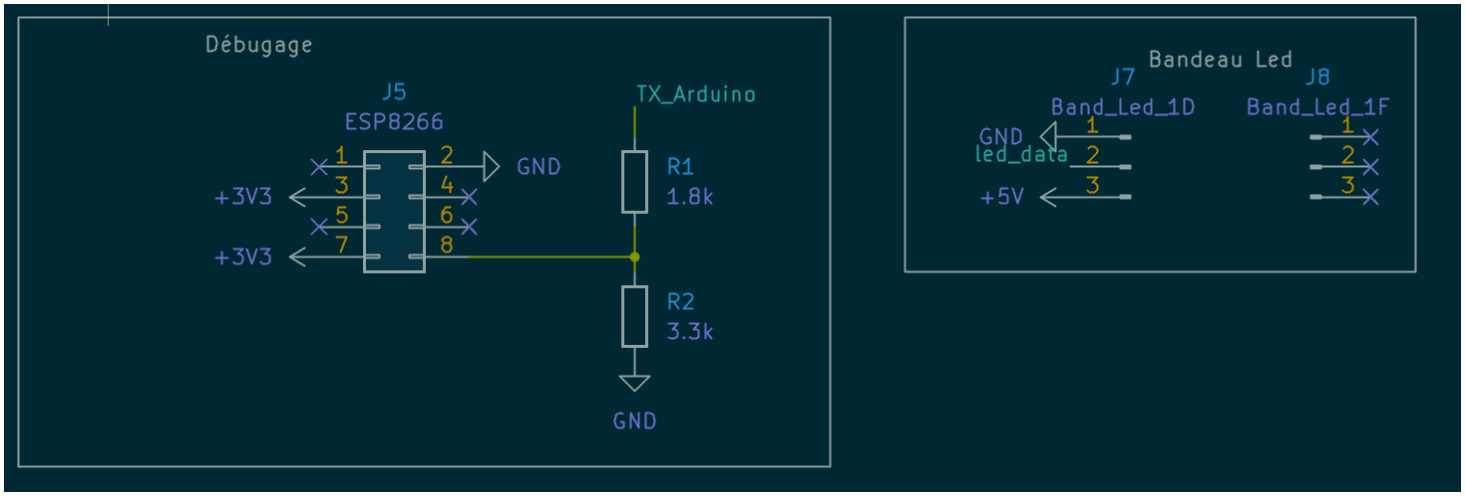
    - **Bandeau LED (J9) :** Un connecteur (Conn_01x03_Pin) est prévu pour l'alimentation et le signal de données (*led_data*) du bandeau LED.

- **Adaptation de Niveau (GT117) :** Ce composant sert d'adaptateur de niveau logique. Il est crucial pour convertir les signaux entre la tension de 5V (typique pour l'Arduino ou certains périphériques) et 3.3V (pour d'autres composants comme les capteurs I2C/SPI mentionnés, y compris potentiellement le MPU-9250 dont les broches d'E/S sont alimentées entre +1,71V et +3,6V).

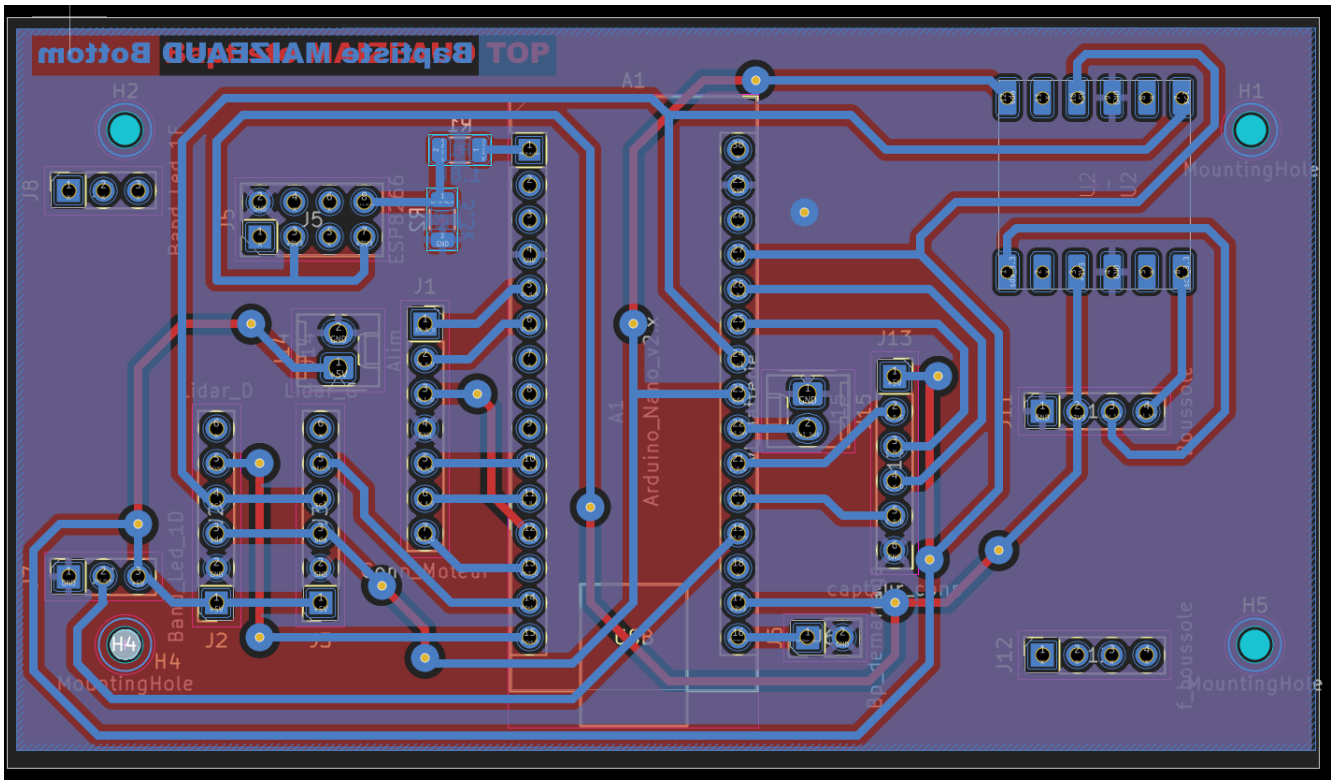




- **Autres Connexions :** Le schéma inclut des connecteurs pour le **Bouton Démarrage** et pour le **Débugage**, ainsi que des symboles d'alimentation (*+5V*, *+3V3*, *GND*) et des résistances (R) pour les circuits annexes.



Routage Kicad



2. IMPLÉMENTATION LOGICIELLE

Code de la Boussole

Ce code utilise la lecture et le traitement des données du magnétomètre intégré au MPU-9250 pour déterminer l'orientation du robot. Après l'initialisation de la communication I2C, il configure le capteur MPU-9250 et applique des corrections de *bias* et d'*échelle* spécifiques au magnétomètre (calibration). Dans la boucle principale, les composantes Y et Z du champ magnétique sont lues. La fonction `atan2` est ensuite utilisée pour calculer l'angle de cap dans le plan Y-Z, fournissant une valeur en degrés affichée via la liaison série. Ce processus est essentiel pour le suivi de CAP

Code de la boussole:

```
#include "MPU9250.h"

MPU9250 mpu;
float val_deg;
void setup() {
  Serial.begin(115200);
  Wire.begin();
  delay(2000);

  if (!mpu.setup(0x68)) {  // change to your own address
    while (1) {
      Serial.println("MPU connection failed. Please check your connection
with `connection_check` example.");
      delay(5000);
    }
  }
  mpu.setMagBias(-1.73, 436.62, -161.37);
  mpu.setMagScale(1.01, 0.94, 1.05);
}

void loop() {
  if (mpu.update()) {
    static uint32_t prev_ms = millis();
    if (millis() - prev_ms >= 1000) {

      // 1. Récupérer les valeurs brutes du magnétomètre pour Y et Z
      float magY = mpu.getMagY();
      float magZ = mpu.getMagZ();
```

```
      // 2. Calculer l'angle avec atan2 (Axe vertical / Axe horizontal du
nouveau plan)
      val_deg = atan2(magZ, magY) * 180.0 / PI;

      // Affichage du résultat
      Serial.print("Angle vertical (Y/Z): ");
      Serial.print(val_deg);
      Serial.println("°");

      prev_ms = millis();
    }
  }
}
```

Code de Test des LED

Ce code est dédié à la validation et à la démonstration fonctionnelle du bandeau LED. Il utilise la librairie `PololuLedStrip` pour contrôler un ruban de 8 LEDs adressables (RGB). Le programme configure un tableau de couleurs (`rgb_color`) qui définit l'état chromatique de chaque diode. Dans la boucle principale, deux séquences de couleurs sont définies et appliquées au ruban via la fonction `rubanLed.write()`, permettant de vérifier l'adressage individuel et la capacité à afficher différentes teintes et luminosités. Un délai de 500ms est inséré pour visualiser clairement le changement d'état.

Code de Test des LED:

```
#include <PololuLedStrip.h>
// Create an ledStrip object and specify the pin it will use.
PololuLedStrip<A0> rubanLed;
// Create a buffer for holding the colors (3 bytes per color).
#define nbLeds 8
rgb_color tableauCouleurs[nbLeds];

void setup()
{

}

void loop() {
```

```
// put your main code here, to run repeatedly:

// on complète le tableau avec les couleurs souhaitées pour chaque led
tableauCouleurs[0] = rgb_color(0,0,10);
tableauCouleurs[1] = rgb_color(10,0,0);
tableauCouleurs[2] = rgb_color(0,10,0);
tableauCouleurs[3] = rgb_color(10,0,10);
tableauCouleurs[4] = rgb_color(0,10,10);
tableauCouleurs[5] = rgb_color(10,10,0);
tableauCouleurs[6] = rgb_color(10,10,10);
tableauCouleurs[7] = rgb_color(2,2,2);
// une fois le tableau complété/modifié,
// on génère le signal pour modifier l'état des leds
rubanLed.write(tableauCouleurs, nbLeds);
delay(500);

// on peut également changer les couleurs indépendamment
tableauCouleurs[0].red=0;
tableauCouleurs[0].green=0;
tableauCouleurs[0].blue=1;

tableauCouleurs[1].red=1;
tableauCouleurs[1].green=0;
tableauCouleurs[1].blue=0;

tableauCouleurs[2].red=0;
tableauCouleurs[2].green=1;
tableauCouleurs[2].blue=0;

tableauCouleurs[3].red=1;
tableauCouleurs[3].green=0;
tableauCouleurs[3].blue=1;

tableauCouleurs[4].red=0;
tableauCouleurs[4].green=1;
tableauCouleurs[4].blue=1;

tableauCouleurs[5].red=1;
tableauCouleurs[5].green=1;
tableauCouleurs[5].blue=0;
```

```
  tableauCouleurs[6].red=1;
  tableauCouleurs[6].green=1;
  tableauCouleurs[6].blue=1;

  tableauCouleurs[7].red=0;
  tableauCouleurs[7].green=0;
  tableauCouleurs[7].blue=0;

  rubanLed.write(tableauCouleurs, nbLeds);
  delay(500);
}
```

Code Principal : Suivi de Cap et Détection d'Obstacles

Ce code constitue le programme embarqué central du robot, structuré autour d'une machine à états finis.

- **Initialisation :** Il configure les registres AVR pour le contrôle direct des E/S des moteurs et initialise le **Timer 1** pour générer des signaux MLI (PWM) à une fréquence de 20 kHz, garantissant une commande précise de la vitesse des moteurs. Il initialise également la communication I2C, configure les capteurs LiDARs pour la détection de distance (mode Long Range) et calibre le MPU-9250.
- **Machine à États et Fonctions :** La navigation du robot est orchestrée par une machine d'état, incluant les fonctions essentielles du code :
    - **États de Navigation Principaux :**
        - etapeBatterie : Gère le démarrage en vérifiant l'état de la batterie (via l'ADC) et en signalant son niveau via les LEDs. Le robot attend l'activation par le bouton de démarrage.
        - etapesuiviligne : État de navigation par défaut, utilisant les capteurs de ligne.
        - etapecap : État de secours qui maintient le cap (via la fonction `suivi_cap()` et MPU-9250) lorsque la ligne est perdue et que le Lidar ne détecte pas d'obstacle proche.
        - etapefrein : État d'urgence déclenché lorsque le Lidar détecte un obstacle à moins de 250 mm, ordonnant l'arrêt immédiat des moteurs.

- **États de Manœuvre :** Les états `tournergauchenoir`, `tournergaucheblanc`, `tournerdroitenoir`, et `tournerdroiteblanc` gèrent les virages précis en réponse à la détection de marques au sol.
- **Fonctions Clés :**
  - `moteurGauche()` et `moteurDroite()` : Gèrent la vitesse des moteurs par MLI (PWM du Timer 1) et leur sens de rotation.
  - `ADC_init()` et `lireADC()` : S'occupent de l'acquisition des données analogiques (batterie et capteurs de ligne).
  - `consigne()` et `suivi_cap()` : Implémentent la boucle de régulation du cap à l'aide du MPU-9250 pour corriger la trajectoire.
  - `mesurerTempsDecharge()` : Utilisée pour la lecture des capteurs de marques au sol.

La boucle principale gère la transition entre ces états

```
#include <Arduino.h>
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <math.h>     // pour log()
#include <Wire.h>     // Communication I2C
#include <VL53L1X.h>  // Bibliothèque du capteur de distance
#include <PololuLedStrip.h>
#include "MPU9250.h"

// --- Variables pour fonctions moteurs ---
int ligne_g = 0;
int ligne_d = 0;
float Kp = 0.2;
int consigne_ligne = 0;
PololuLedStrip<A0> rubanLed;
#define nbLeds 8
rgb_color tableauCouleurs[nbLeds];  // tableau des couleurs des LEDs

// --- TABLEAUX DE SCÉNARIO ---

uint8_t tab_gauche[] = { 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1 };
```

```
// Index :                0, 1, 2, 3, 4, 5, 6, 7, 8,  9, 10
uint8_t tab_droite[] = { 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0 };

MPU9250 mpu;

const int xshut1 = 11;  // Capteur 1
const int xshut2 = 12;  // Capteur 2
VL53L1X lidar1;         // Premier capteur
VL53L1X lidar2;
int distance1;
volatile float consigne_cap;

// Capteurs de Marque
int tempsChargeA1 = 0;
int tempsChargeA2 = 0;
int A1_Precedent = 0;
int A2_Precedent = 0;
const int SEUIL_temps_Charge_Noir = 27500;  // tempsChargeNoir = 30000
uint8_t Droite = 0;
uint8_t Gauche = 0;

// Machine d'état
enum state {
  etapeBatterie,
  etapesuiviligne,
  etapecap,
  tournergauchenoir,
  tournergaucheblanc,
  tournerdroitenoir,
  tournerdroiteblanc,
  etapefrein
};
state etapeActive = etapeBatterie;
state etapeSuivante = etapeBatterie;

// --- Déclarations des fonctions des autres onglets (Prototypes) ---
void ADC_init();
uint16_t lireADC(uint8_t canal);
```

```
void consigne();
void suivi_cap();
unsigned long mesurerTempsDecharge(uint8_t pinBit);
void moteurGauche(int vitesse, bool avance, bool recule);
void moteurDroite(int vitesse, bool avance, bool recule);

// --- Programme Principal ---
int main() {

  // 1. DIRECTION DES BROCHES (Sorties moteurs)
  DDRB |= (1 << PB0) | (1 << PB1) | (1 << PB2);
  DDRD |= (1 << PD7) | (1 << PD3) | (1 << PD2);

  // 2. CONFIGURATION STRICTE DU TIMER 1 À 20 kHz
  TCCR1A = 0;
  TCCR1B = 0;
  TCCR1A |= (1 << WGM11) | (1 << COM1A1) | (1 << COM1B1);
  TCCR1B |= (1 << WGM13) | (1 << WGM12);
  ICR1 = 799;
  TCCR1B |= (1 << CS10);
  OCR1A = 0;
  OCR1B = 0;
  PORTD &= ~((1 << PD7) | (1 << PD3) | (1 << PD2));
  PORTB &= ~(1 << PB0);

  // 3. ARDUINO INITIALISATION
  init();

  DDRC &= ~((1 << PC1) | (1 << PC2));

  // --- CONFIGURATION DU BOUTON PB5 ---
  DDRB |= (1 << PB5);
  PORTB |= (1 << PB5);

  // --- INITIALISATION LIDARS ---
  ADC_init();
  Wire.begin();
  Wire.setClock(400000);
  pinMode(xshut1, OUTPUT);
  pinMode(xshut2, OUTPUT);
```

```
digitalWrite(xshut1, LOW);
digitalWrite(xshut2, LOW);
delay(100);
digitalWrite(xshut1, HIGH);
delay(100);
lidar1.setTimeout(500);
if (!lidar1.init()) {
  while (1)
    ;
}
lidar1.setAddress(0x2A);
lidar1.setDistanceMode(VL53L1X::Long);
lidar1.startContinuous(50);
digitalWrite(xshut2, HIGH);
delay(100);
lidar2.setTimeout(500);
if (!lidar2.init()) {
  while (1)
    ;
}
lidar2.setDistanceMode(VL53L1X::Long);
lidar2.startContinuous(50);
sei();

TCCR0A = (1 << WGM01) | (1 << WGM00);
TCCR0B = (1 << CS01) | (1 << CS00);
TIMSK0 = (1 << TOIE0);

int distance1 = 500;
Serial.begin(115200);
Wire.begin();

if (!mpu.setup(0x68)) {
  while (1) {
    Serial.println("MPU connection failed.");
    delay(5000);
  }
}

// Calibration magnétomètre
```

```
mpu.setMagBias(-26.01, 302.68, -121.03);
mpu.setMagScale(0.99, 1.02, 0.99);
consigne();

// 4. BOUCLE PRINCIPALE
while (1) {
  // Lecture unique centralisée
  ligne_g = analogRead(A6);
  ligne_d = analogRead(A7);
  mpu.update_mag();

  // Diagnostic visuel sur le ruban LED
  if (ligne_d > 150) {
    tableauCouleurs[0] = rgb_color(100, 0, 0);
  } else {
    tableauCouleurs[0] = rgb_color(0, 0, 0);
  }
  if (ligne_g > 150) {
    tableauCouleurs[1] = rgb_color(100, 0, 0);
  } else {
    tableauCouleurs[1] = rgb_color(0, 0, 0);
  }
  rubanLed.write(tableauCouleurs, nbLeds);

  // Lecture du LiDAR non-bloquante
  if (lidar1.dataReady()) {
    distance1 = lidar1.read();
  }

  // Gestion globale du bouton pour arrêter le robot à tout moment
  if (etapeSuivante == etapecap && !(PINB & (1 << PB5))) {
    moteurGauche(0, false, false);
    moteurDroite(0, false, false);
    etapeSuivante = etapeBatterie;
    _delay_ms(300);
  }

  switch (etapeActive) {
    case etapeBatterie:
      {
```

```
          moteurGauche(0, false, false);
          moteurDroite(0, false, false);
          float batt = lireADC(3) * (5.07 / 1023.0);
          if (batt < 3.0) {
            for (int i = 0; i < nbLeds; i++) tableauCouleurs[i] =
rgb_color(10, 0, 0);
          } else if (batt < 3.5) {
            for (int i = 0; i < nbLeds; i++) tableauCouleurs[i] =
rgb_color(10, 3, 0);
          } else {
            for (int i = 0; i < nbLeds; i++) tableauCouleurs[i] =
rgb_color(0, 10, 0);
          }
          rubanLed.write(tableauCouleurs, nbLeds);

          if (!(PINB & (1 << PB5))) {
            _delay_ms(300);
            etapeSuivante = etapesuiviligne;
          }
        }
        break;

      case etapesuiviligne:
        for (int i = 0; i < nbLeds; i++) tableauCouleurs[i] = rgb_color(0,
20, 0);
        rubanLed.write(tableauCouleurs, nbLeds);

        if (distance1 >= 250) {
          tempsChargeA1 = mesurerTempsDecharge(PC1);
          tempsChargeA2 = mesurerTempsDecharge(PC2);

          // --- LOGIQUE DES MARQUES CORRIGÉE (FRONT MONTANT) ---

          if (tempsChargeA1 > SEUIL_temps_Charge_Noir && A1_Precedent <
SEUIL_temps_Charge_Noir) {
            Droite = Droite + 1;
            // Évaluation de la décision uniquement à l'instant où la
marque est franchie
            if (Droite <= 11 && tab_droite[(Droite - 1)] == 1) {
              etapeSuivante = tournerdroitenoir;
```

```
            }
          }

          if (tempsChargeA2 > SEUIL_temps_Charge_Noir && A2_Precedent <
SEUIL_temps_Charge_Noir) {
            Gauche = Gauche + 1;
            // Évaluation de la décision uniquement à l'instant où la
marque est franchie
            if (Gauche <= 11 && tab_gauche[(Gauche - 1)] == 1) {
              etapeSuivante = tournergauchenoir;
            }
          }

          A1_Precedent = tempsChargeA1;
          A2_Precedent = tempsChargeA2;

          // Perte de ligne (fin de parcours ou ligne pointillée)
          if (ligne_g < 50 && ligne_d < 50) {
            moteurGauche(0, true, true);
            moteurDroite(0, true, true);
            consigne();
            etapeSuivante = etapecap;
          }


          if (etapeSuivante == etapesuiviligne && (ligne_g > 50 || ligne_d
> 50)) {
            int v = 275;
            float erreur = (ligne_g - ligne_d) - consigne_ligne;
            float action = (Kp * erreur);

            int vitesseG = v + action;
            int vitesseD = v - action;

            if (vitesseG > 799) vitesseG = 799;
            if (vitesseG < 0) vitesseG = 0;
            if (vitesseD > 799) vitesseD = 799;
            if (vitesseD < 0) vitesseD = 0;

            moteurGauche(vitesseG, true, false);
```

```
          moteurDroite(vitesseD, true, false);
        }
      } else {
        etapeSuivante = etapefrein;
      }
      break;

    case etapefrein:
      moteurGauche(0, true, true);
      moteurDroite(0, true, true);
      break;

    case tournergauchenoir:
      moteurGauche(390, false, true);
      moteurDroite(390, true, false);
      if (ligne_g < 100 && ligne_d < 100) {
        etapeSuivante = tournergaucheblanc;
        _delay_ms(20);
      }
      break;

    case tournergaucheblanc:
      moteurGauche(350, false, true);
      moteurDroite(350, true, false);
      if (ligne_g > 150 && ligne_d > 150) {
        moteurGauche(0, false, true);
        moteurDroite(0, true, false);
        A1_Precedent = 0;
        A2_Precedent = 0;
        // Conservation de Gauche en mémoire pour continuer le scénario
        etapeSuivante = etapesuiviligne;
        _delay_ms(5);
      }
      break;

    case tournerdroitenoir:
      moteurGauche(390, true, false);
      moteurDroite(390, false, true);
      if (ligne_g < 100 && ligne_d < 100) {
        etapeSuivante = tournerdroiteblanc;
```

```
          _delay_ms(20);
        }
        break;

      case tournerdroiteblanc:
        moteurGauche(350, true, false);
        moteurDroite(350, false, true);
        if (ligne_g > 150 && ligne_d > 150) {
          moteurGauche(0, false, true);
          moteurDroite(0, true, false);
          A1_Precedent = 0;
          A2_Precedent = 0;
          // Conservation de Droite en mémoire pour continuer le scénario
          etapeSuivante = etapesuiviligne;
          _delay_ms(5);
        }
        break;

      case etapecap:
        for (int i = 0; i < nbLeds; i++) tableauCouleurs[i] = rgb_color(0,
20, 20);
        rubanLed.write(tableauCouleurs, nbLeds);
        consigne();
        if (ligne_g >= 100 || ligne_d >= 100) {
          moteurGauche(0, true, true);
          moteurDroite(0, true, true);
          etapeSuivante = etapesuiviligne;
        }

        if (distance1 >= 250) {
          if (mpu.available()) {
            mpu.update_mag();
            suivi_cap();
          }
        } else {
          etapeSuivante = etapefrein;
        }
        break;
    }
    etapeActive = etapeSuivante;
```

```
  }
}
```

Code de Contrôle des Moteurs

Ces fonctions utilisent trois paramètres : la **vitesse** (qui est la valeur de rapport cyclique pour l'OCR du Timer 1), et deux booléens pour la **direction** (`avance`, `recule`). Le contrôle de la direction est réalisé par l'activation/désactivation de bits spécifiques sur les ports E/S (PORTD et PORTB). La vitesse est définie en écrivant directement dans les registres de comparaison du Timer 1 (`OCR1A` et `OCR1B`), ce qui module la largeur d'impulsion (MLI) appliquée aux ponts en H des moteurs.

Code moteur:

```
// --- Fonctions de contrôle des moteurs ---
void moteurGauche(int vitesse, bool avance, bool recule) {
  if (avance) PORTD |= (1 << PD7);
  else PORTD &= ~(1 << PD7);


  if (recule) PORTB |= (1 << PB0);
  else PORTB &= ~(1 << PB0);


  OCR1A = vitesse;
}


void moteurDroite(int vitesse, bool avance, bool recule) {
  if (avance) PORTD |= (1 << PD2);
  else PORTD &= ~(1 << PD2);


  if (recule) PORTD |= (1 << PD3);
  else PORTD &= ~(1 << PD3);


  OCR1B = vitesse;
```

```
}
```

Code de Gestion du Cap (MPU)

Cette partie contient les fonctions de navigation cruciales utilisant le MPU-9250.

- `consigne()` : Initialise la consigne de cap. Elle lit les données du magnétomètre (Y et Z) au démarrage, calcule l'angle de cap initial via `atan2` et définit la valeur cible (`consigne_cap`) que le robot doit maintenir.
- `suivi_cap()` : Implémente la boucle de régulation du cap. Elle calcule l'erreur angulaire entre le cap actuel et la consigne. Cette erreur est utilisée pour moduler la vitesse différentielle (`vr`) appliquée aux moteurs : plus l'erreur est grande, plus la correction de vitesse est importante, permettant au robot de corriger sa trajectoire pour revenir au cap défini.

Code MPU

```
void consigne() {

  for (int i=0; i<=2; i++){
  mpu.update_mag();
  mpu.update_mag();
  float hz = mpu.getMagZ();
  float hy = mpu.getMagY();
  consigne_cap = (atan2(hz, hy) * 180.0 / M_PI);
  }

  return consigne_cap;
}
void suivi_cap() {
  float v = 260.0;
  float vr = 150;

  float hz = mpu.getMagZ();
  float hy = mpu.getMagY();
  float angle = atan2(hz, hy) * (180.0 / M_PI);
  float erreur = abs(consigne_cap - angle);
  vr = vr * (erreur / 180);
  if (consigne_cap < angle) {
    moteurDroite((v + vr), true, false);
    moteurGauche((v - vr), true, false);
```

```
  }
  if (consigne_cap > angle) {
    moteurDroite((v - vr), true, false);
    moteurGauche((v + vr), true, false);
  }
}
```

Code d'Acquisition Analogique (ADC)

Ce code gère l'interface avec le convertisseur analogique-numérique (ADC) du microcontrôleur, notamment pour la lecture de la tension de la batterie.

- `ADC_init()` : Configure les registres de l'ADC, spécifiant la référence de tension (AVcc) et activant l'ADC avec un *prescaler* de 128.
- `lireADC(uint8_t canal)` : Fonction de lecture synchrone qui prend le canal analogique en paramètre. Elle sélectionne le multiplexeur (ADMUX), lance la conversion (`ADSC`) et attend passivement (boucle de *polling*) que la conversion soit terminée avant de retourner la valeur numérique sur 10 bits.

Code adc

```
void ADC_init() {
  // Référence AVcc (5V)
  ADMUX = (1 << REFS0);
  // Activation ADC + prescaler 128
  ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
}



uint16_t lireADC(uint8_t canal) {
  ADMUX = (ADMUX & 0xF0) | (canal & 0x0F);
  ADCSRA |= (1 << ADSC);
  while (ADCSRA & (1 << ADSC))
    ;
  return ADC;
}
```

3 : Arda (CNY)

```
#include <Arduino.h>
const uint8_t capteurPin = 6; // PD5
unsigned long tempsCharg;
int main(void) {
 init();
 Serial.begin(9600);
 DDRD |= (1 << PD6);
 DDRD &= ~(1 << PD6);
 while (1) {
   DDRD |= (1 << PD6);         // OUTPUT
   PORTD |= (1 << PD6);      // HIGH
   delayMicroseconds(50);
   DDRD &= ~(1 << PD6);        // INPUT
   unsigned long debut = micros();
   while ((PIND & (1 << PIND6)) &&
          (micros() - debut < 30000)) {
   }
   tempsCharg = micros() - debut;
   if (tempsCharg > 100) {
     Serial.println("Surface : Noir");
   } else {
     Serial.println("Surface : Blanc");
```

```
    }
    Serial.print("Temps : ");
    Serial.print(tempsCharg);
    Serial.println(" µs");
    delay(200);
  }
}
/*
     DOUBLE CAPTEUR VL53L1X
Objectif :
- Utiliser 2 capteurs de distance (LiDAR)
- Les faire fonctionner ensemble sur le bus I2C
- Lire les distances en millimètres
Principe important :
Les deux capteurs ont la même adresse I2C au départ (0x29)
Donc on utilise les broches XSHUT pour les activer un par un
et changer l'adresse du premier capteur.
*/
#include <avr/io.h>         // Accès bas niveau au microcontrôleur
#include <util/delay.h>     // Fonctions de temporisation
#include <Wire.h>           // Communication I2C
#include <VL53L1X.h>        // Bibliothèque du capteur de distance
// ---------- Broches XSHUT ----------
```

```
// XSHUT permet d'allumer / éteindre un capteur LiDAR
const int xshut1 = 11;     // Capteur 1
const int xshut2 = 12;     // Capteur 2
// ---------- Déclaration des capteurs ----------
VL53L1X lidar1;            // Premier capteur
VL53L1X lidar2;            // Deuxième capteur
int main()
{
   init(); // Initialise Arduino
   // INITIALISATION COMMUNICATION PC (SERIAL)
   Serial.begin(115200);   // Communication avec le PC
   // INITIALISATION BUS I2C
   Wire.begin();           // Démarre le bus I2C
   Wire.setClock(400000);  // Vitesse rapide (400 kHz)
   // CONFIGURATION DES BROCHES XSHUT
   pinMode(xshut1, OUTPUT);   // Broche capteur 1 en sortie
   pinMode(xshut2, OUTPUT);   // Broche capteur 2 en sortie
   // On coupe les deux capteurs au démarrage
   digitalWrite(xshut1, LOW);
   digitalWrite(xshut2, LOW);
   delay(100); // Temps pour bien tout réinitialiser
   // INITIALISATION CAPTEUR 1
   digitalWrite(xshut1, HIGH); // On active seulement capteur 1
```

```
delay(100);                  // Temps de démarrage capteur
lidar1.setTimeout(500);      // Si pas de réponse -> erreur
if (!lidar1.init())          // Test si capteur détecté
{
    Serial.println("Erreur lidar 1");
    while (1);               // Bloque le programme si erreur
}
lidar1.setAddress(0x2A);     // On change son adresse I2C
lidar1.setDistanceMode(VL53L1X::Long); // Mode longue distance
lidar1.startContinuous(50);  // Mesure automatique toutes les 50 ms
// INITIALISATION CAPTEUR 2
digitalWrite(xshut2, HIGH);  // On active capteur 2
delay(100);                  // Temps de démarrage
lidar2.setTimeout(500);      // Timeout de sécurité
if (!lidar2.init())          // Test capteur 2
{
    Serial.println("Erreur lidar 2");
    while (1);               // Stop si erreur
}
// Le capteur 2 garde l'adresse par défaut 0x29
lidar2.setDistanceMode(VL53L1X::Long); // Même mode
lidar2.startContinuous(50);  // Mesure automatique
Serial.println("Deux lidars prêts");
```

```
    // BOUCLE PRINCIPALE

    while (1)

    {

        // Lecture distance capteur 1 (en mm)

        int distance1 = lidar1.read();

        // Lecture distance capteur 2 (en mm)

        int distance2 = lidar2.read();

        // Affichage des distances sur le moniteur série

        Serial.print("Lidar 1 : ");

        Serial.println(distance1);

        Serial.print(" mm   ");

        Serial.println("Lidar 2 : ");

        Serial.print(distance2);

        Serial.println(" mm");

        delay(100); // Petite pause pour éviter surcharge

    }

}


#include <Wire.h>
#include <VL53L1X.h>
#include <PololuLedStrip.h>
PololuLedStrip<A0> rubanLed;
#define nbLeds 8
rgb_color tableauCouleurs[nbLeds]; // tableau des couleurs des LEDs
VL53L1X lidar1;
VL53L1X lidar2;
// Réglages
```

```
int Kp = 3; // Gain proportionnel
int consigne = 200; // Distance cible (en mm)
int vitesseBase = 100; // Vitesse de base des moteurs
int Vmax = 150;
int Vmin = 0;
int seuilNoir = 900; // seuil détection sol noir (capteurs analogiques)
int erreur, action, mesure1,mesure2,moteurGauche, moteurDroit;
const int XSHUT1 = PB3; // broches XSHUT des LIDAR
const int XSHUT2 = PB4; // broches XSHUT des LIDAR
enum state {
 etapeInit, // état initial (attente)
 etapeSuiviDistance, // suivi de distance avec régulation
 etapeArretNoir // arrêt si ligne noire détectée
};
state etapeActive = etapeInit;
state etapeSuivante = etapeInit;
void ADC_init()
{
 // Référence AVcc
 ADMUX = (1 << REFS0); // référence tension = AVcc (5V)
 // Activation ADC + prescaler 128
 ADCSRA = (1 << ADEN) |(1 << ADPS2) |(1 << ADPS1) |(1 << ADPS0);
}
uint16_t lireADC(uint8_t canal) // lecture d'un canal analogique (A0-A7)
{
 ADMUX = (ADMUX & 0xF0) | (canal & 0x0F); // sélection canal
 ADCSRA |= (1 << ADSC); // démarrage conversion
 while (ADCSRA & (1 << ADSC)); // attente fin conversion
 return ADC; // valeur 0-1023
}
void initFonctionsTempsArduino()
{
 sei(); // activation interruptions globales
 TCCR0A = (1 << WGM01) | (1 << WGM00); // configuration Timer0 en PWM
rapide
 TCCR0B = (1 << CS01) | (1 << CS00);
 TIMSK0 = (1 << TOIE0); // interruption overflow
}
void setVitesse(int gauche, int droite)
{
```

```
 if (gauche > Vmax) gauche = Vmax; // saturation des vitesses
 if (gauche < Vmin) gauche = Vmin;
 if (droite > Vmax) droite = Vmax;
 if (droite < Vmin) droite = Vmin;
 OCR1B = gauche; // écriture PWM moteurs (Timer1)
 OCR1A = droite;
}
int main()
{
 init();  // init Arduino bas niveau
 ADC_init(); // init convertisseur analogique
 initFonctionsTempsArduino();
 sei(); // activation interruptions
 // I2C + capteur (bloc 1)
 Wire.begin(); // Démarrage bus I2C
 Wire.setClock(400000); // I2C rapide (400 kHz)
 Serial.begin(115200); // debug
 DDRB |= (1 << XSHUT1); // activation broches XSHUT LIDAR
 DDRB |= (1 << XSHUT2); // activation broches XSHUT LIDAR
 PORTB &= ~(1 << XSHUT1); // on reset les capteurs au départ
 PORTB &= ~(1 << XSHUT2); // on reset les capteurs au départ
 delay(10);
 PORTB |= (1 << XSHUT1);  // activation LIDAR 1
 lidar1.setTimeout(500); // Timeout lecture capteur (ms)
 if (!lidar1.init()) // Vérification du capteur
 {
   Serial.println("Failed to detect and initialize sensor!");
   while (1); // Blocage si capteur absent
 }
 lidar1.setAddress(0x2A); // changement adresse I2C
 lidar1.setDistanceMode(VL53L1X::Long); // Mode longue portée
 lidar1.setMeasurementTimingBudget(50000); // Temps de mesure (µs)
 lidar1.startContinuous(50); // Mesure continue toutes les 50 ms
 PORTB |= (1 << XSHUT2); // activation LIDAR 2
 lidar2.setTimeout(500);
 if (!lidar2.init())
 {
   Serial.println("Erreur lidar 2");
   while (1);
 }
```

```
 lidar2.setDistanceMode(VL53L1X::Long);
 lidar2.setMeasurementTimingBudget(50000);
 lidar2.startContinuous(50);
 // moteurs
 DDRD |= (1 << PD2) | (1 << PD3) | (1 << PD7); // Direction moteurs
 DDRB |= (1 << PB0) | (1 << PB1) | (1 << PB2) | (1 << PB5) ; // PWM
moteurs et bouton
 DDRC &= ~(1 << PC0); // entrée analogique
 DDRC |= (1 << PC0);
 TCCR1A = (1 << COM1A1) | (1 << COM1B1) | (1 << WGM10); // Configuration
PWM Timer1
 TCCR1B = (1 << WGM12) | (1 << CS11);
 PORTD |=  (1 << PD2); // sens moteur gauche
 PORTD &= ~(1 << PD3); // sens inverse désactivé
 PORTD |=  (1 << PD7); // sens moteur droit
 PORTB &= ~(1 << PB0); // sens inverse désactivé;
 PORTB |= (1 << PB5);  //pull-up activée
 while (1)
 {
   bool surNoir = (lireADC(6) > seuilNoir || lireADC(7) > seuilNoir); //
détection ligne noire (capteurs analogiques)
   switch (etapeActive)
     {
       case etapeInit:
     {
     //BATTERIE
     float batt = lireADC(3) * (5.07 / 1023);
     if (batt < 3.0)
     {
       for (int i = 0; i < nbLeds; i++)
           tableauCouleurs[i] = rgb_color(10, 0, 0); // rouge
       etapeSuivante = etapeInit; // blocage
       setVitesse(0, 0);
       break;
     }
     else if (batt < 3.1)
     {
       for (int i = 0; i < nbLeds; i++)
           tableauCouleurs[i] = rgb_color(10, 3, 0); // orange
     }
```

```
      else
      {
        for (int i = 0; i < nbLeds; i++)
            tableauCouleurs[i] = rgb_color(0, 10, 0); // vert
      }
      rubanLed.write(tableauCouleurs, nbLeds);
      //BOUTON
      if (!(PINB & (1 << PB5)))
      {
        etapeActive = etapeSuivante;
        etapeSuivante = etapeSuiviDistance;
      }
      else
      {
        setVitesse(0, 0);
        etapeSuivante = etapeInit;
      }
      break;
      }
      case etapeSuiviDistance:
        if (surNoir) {
          etapeActive = etapeSuivante;
          etapeSuivante = etapeArretNoir;
        }
        else {
           mesure2 = lidar2.read();  // lecture distance LIDAR
           erreur = consigne - mesure2;  // calcul erreur de distance
           action = (Kp * erreur) / 10; // correcteur proportionnel
           moteurGauche = vitesseBase - action; // calcul vitesses moteurs
différentielles
           moteurDroit  = vitesseBase + action; // calcul vitesses moteurs
différentielles
          if (action > Vmax)  // saturation
            {
            action = Vmax;
            }
            if (action < Vmin)
            {
              action = Vmin;
            }
```

```
        setVitesse(moteurGauche, moteurDroit);
      }
      break;
    case etapeArretNoir:
      setVitesse(0, 0);
      break;
  }
}
// UPDATE ETAT
etapeActive = etapeSuivante; // mise à jour état automate
// Serial.print("A6=");
// Serial.print(capteurMilieuG);
// Serial.print(" A7=");
// Serial.print(capteurMilieuD);
//Debug (optionnel)
//Serial.print("mesure: ");
//Serial.println(mesure2);
// Serial.print(" erreur: ");
// Serial.println(erreur);
// Serial.print(sensor.read());
// if (sensor.timeoutOccurred()) { Serial.print(" TIMEOUT"); }
// Serial.println();}
```

Développement du robot autonome

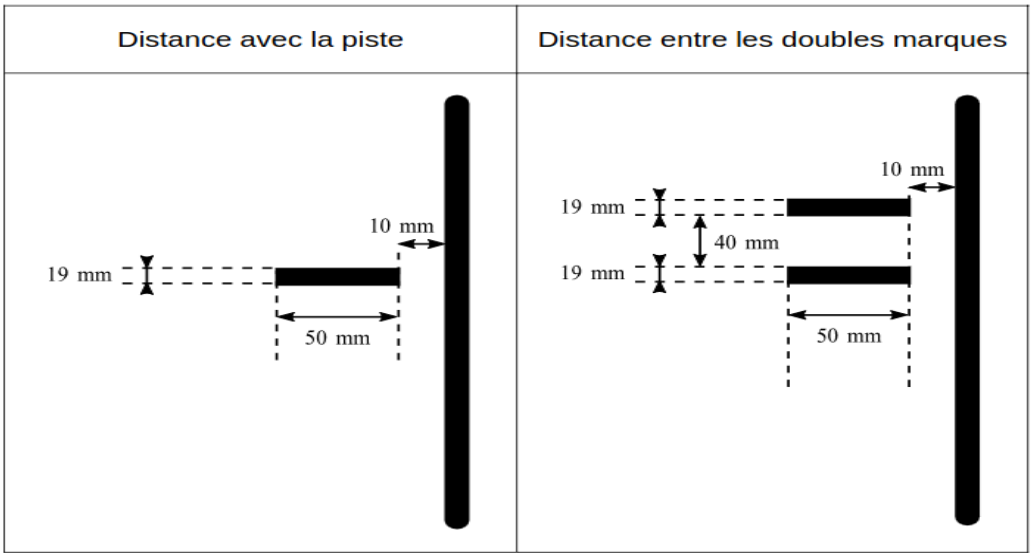
Introduction

Ce projet de SAÉ avait pour objectif la conception d'un robot autonome capable de se déplacer sur un parcours en suivant une ligne noire, en détectant différentes marques et en assurant un suivi de mur à l'aide de capteurs de distance. L'ensemble du système repose sur une architecture embarquée comprenant des capteurs, une carte électronique réalisée sous KiCad et un programme développé sur Arduino Nano.

Le travail a été réalisé de manière progressive en suivant une logique proche d'un vrai projet d'ingénierie. On commence par l'étude et les tests des capteurs, puis on passe à la conception et la fabrication de la carte

électronique. Ensuite vient l'étude des capteurs de distance et enfin l'intégration du programme global du robot.

Article 7 : Caractéristiques des marques

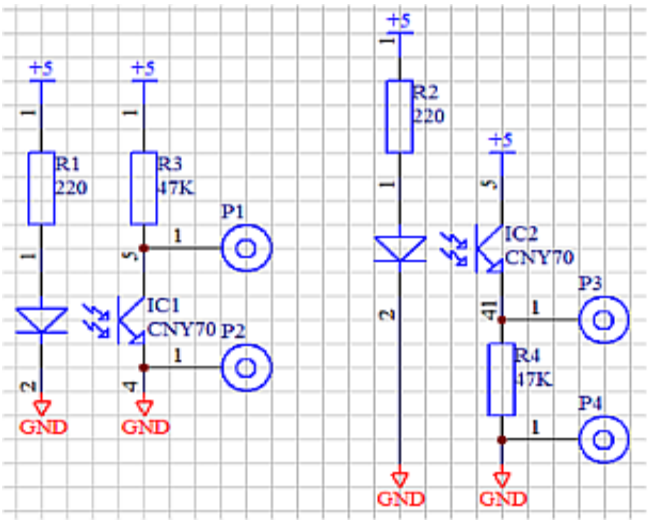


Les marques sont placées à une distance de 25 cm minimum des différentes intersections et actions.

Étude des capteurs de ligne CNY70

La première étape du projet a consisté à comprendre et tester les capteurs de ligne CNY70. Ces capteurs sont essentiels pour permettre au robot de détecter une ligne noire sur un fond clair et ainsi corriger sa trajectoire en permanence.

Le fonctionnement repose sur une émission infrarouge et une réception de la lumière réfléchie. La LED infrarouge éclaire le sol et le phototransistor récupère la lumière renvoyée par la surface. Une surface blanche réfléchit fortement alors qu'une surface noire absorbe la lumière, ce qui crée une différence de signal exploitable.

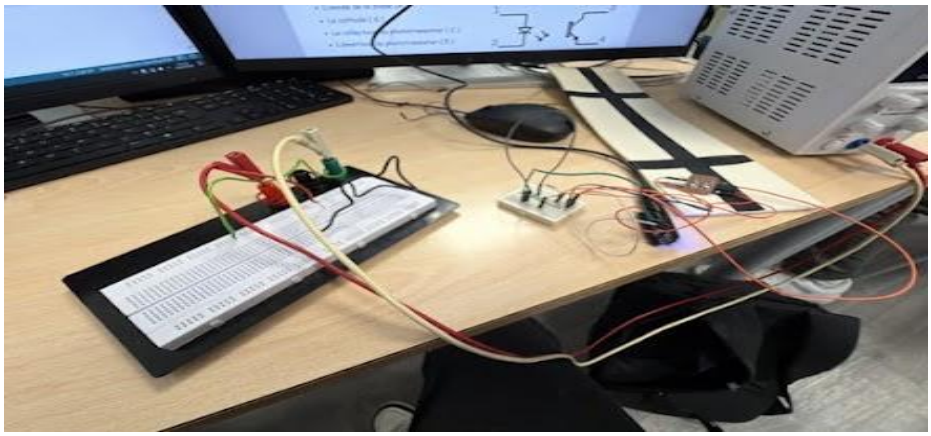


La présence d'un objet provoque la conduction du transistor.

	Etat de la sortie (V)	
Montage	Gauche	Droit
Présence Objet	<5V	<5V
Absence Objet	5V	0V

Avant toute intégration sur le robot, des tests ont été réalisés sur table afin d'observer les valeurs obtenues sur différentes surfaces. Le montage expérimental comprend une résistance de 220 ohms pour la LED et une résistance de 15 kilo ohms pour le phototransistor afin de former un pont diviseur de tension.

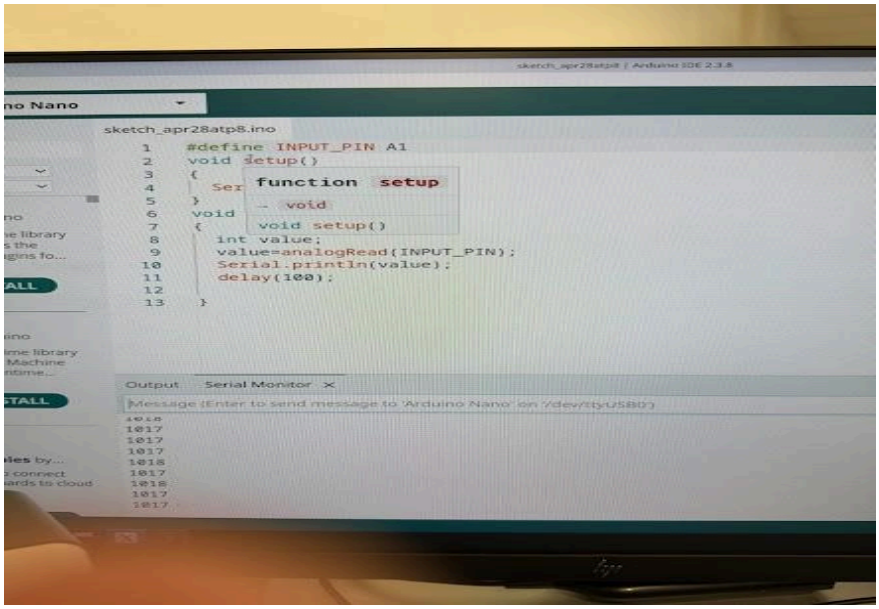
Ces essais ont permis de constater une différence importante entre les valeurs mesurées sur le blanc et sur le noir. Cette différence a permis de définir un seuil de détection autour de 900, utilisé ensuite dans le programme principal.



Le code de test permet de vérifier simplement la détection de la ligne à partir des valeurs analogiques.

```
int capteurG = analogRead(A6);

int capteurD = analogRead(A7);
```

```
int seuil = 500;

if (capteurG > seuil || capteurD > seuil)

{

  Serial.println("ligne détectée");

} else

{

  Serial.println("fond blanc");

}
```



Dans la configuration finale, deux capteurs analogiques sont utilisés pour assurer un suivi continu de la ligne. Ils permettent au robot de rester centré en ajustant en permanence la vitesse des moteurs.

En complément, deux capteurs numériques ont été utilisés afin de détecter les marques présentes sur le parcours. Ces marques servent à déclencher des actions particulières comme des changements de direction ou des arrêts.

Le fonctionnement de ces capteurs numériques repose sur une mesure de temps de charge d’un circuit RC. Le microcontrôleur passe la broche en

sortie pour charger le condensateur puis en entrée pour mesurer le temps de décharge. Ce temps varie selon la réflexion de la surface.

```
#include <Arduino.h>

const uint8_t capteurPin = 6; // PD5

unsigned long tempsCharg;

int main(void) {

 init();

 Serial.begin(9600);

 DDRD |= (1 << PD6);

 DDRD &= ~(1 << PD6);

 while (1) {

   DDRD |= (1 << PD6);         // OUTPUT

   PORTD |= (1 << PD6);      // HIGH

   delayMicroseconds(50);

   DDRD &= ~(1 << PD6);        // INPUT

   unsigned long debut = micros();

   while ((PIND & (1 << PIND6)) &&

          (micros() - debut < 30000)) {

   }

   tempsCharg = micros() - debut;

   if (tempsCharg > 100) {

     Serial.println("Surface : Noir");

   } else {

     Serial.println("Surface : Blanc");
```

```
    }
    Serial.print("Temps : ");
    Serial.print(tempsCharg);
    Serial.println(" µs");
    delay(200);
  }
}
```

Conception électronique et fabrication de la carte

Une fois les capteurs de ligne validés, la conception de la carte électronique a été réalisée sous KiCad. Cette étape est essentielle car elle permet de regrouper tous les composants du robot sur un seul support compact et fiable.

Le schéma électronique a été conçu en premier lieu. Il regroupe les capteurs de ligne, les capteurs de distance, les drivers moteurs ainsi que l'alimentation et les connexions vers l'Arduino Nano. Cette étape permet de vérifier que toutes les connexions sont correctes avant de passer au routage.

Le routage consiste ensuite à transformer le schéma en circuit imprimé. Les pistes sont tracées afin de relier tous les composants entre eux. Une attention particulière a été portée à la séparation entre les signaux de puissance et les signaux logiques afin de limiter les perturbations électriques.

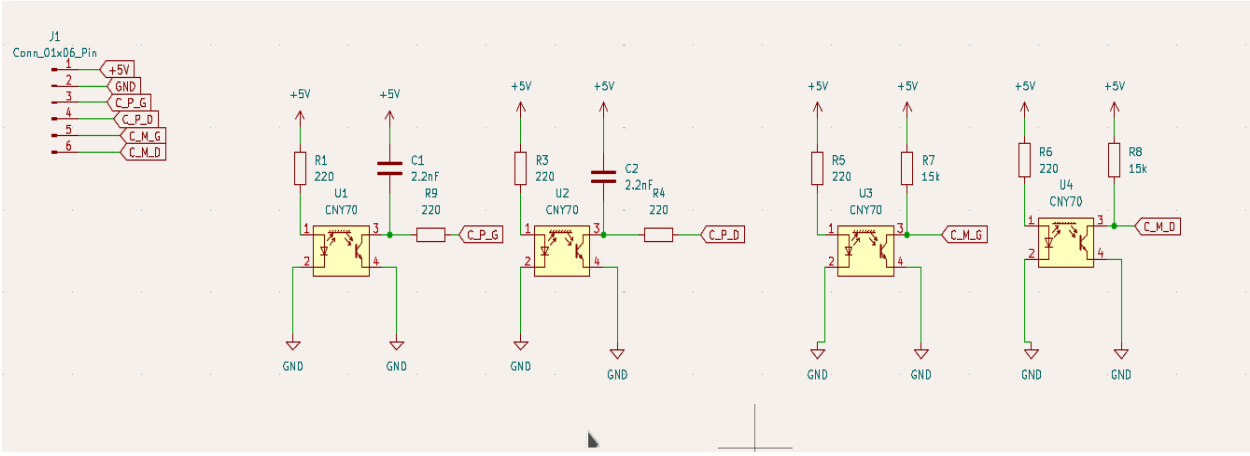
Les pistes d'alimentation des moteurs ont été élargies afin de supporter les courants nécessaires. Les composants ont également été placés de manière à réduire la longueur des pistes et à améliorer la lisibilité du circuit.

Une fois le routage validé, les fichiers Gerber ont été générés pour la fabrication de la carte.

La fabrication s’est déroulée en plusieurs étapes. Les fichiers ont été envoyés pour produire le circuit imprimé. Une fois la carte reçue, les composants ont été soudés manuellement. Cette étape demande de la précision afin d’éviter les faux contacts ou les erreurs de soudure.

Les composants assemblés comprennent les résistances, les connecteurs, les capteurs et les drivers moteurs. Une fois le montage terminé, plusieurs tests ont été réalisés afin de vérifier le bon fonctionnement de la carte.

Un premier test de continuité a permis de vérifier l’absence de court-circuit. Ensuite la carte a été alimentée progressivement afin de contrôler les différents blocs. Enfin les capteurs ont été testés directement sur la carte pour valider leur intégration.



Étude des capteurs LiDAR VL53L1X

Après la validation de la partie ligne, le projet s’est concentré sur les capteurs de distance VL53L1X. Ces capteurs sont utilisés pour le suivi de mur et permettent de mesurer avec précision la distance entre le robot et un obstacle.

Le fonctionnement repose sur la technologie Time Of Flight. Le capteur envoie une impulsion lumineuse infrarouge et mesure le temps nécessaire

pour que cette lumière revienne après réflexion sur un obstacle. Cette mesure est ensuite convertie en distance en millimètres.

Deux capteurs sont utilisés afin de pouvoir mesurer la distance des deux côtés du robot. Cependant, ces capteurs possèdent la même adresse I2C par défaut, ce qui empêche leur utilisation simultanée sans configuration particulière.

Pour résoudre ce problème, les broches XSHUT sont utilisées. Elles permettent d'activer les capteurs un par un au démarrage afin de leur attribuer une adresse différente. Une fois cette configuration réalisée, les deux capteurs peuvent fonctionner ensemble sur le même bus I2C sans conflit.

```
#include <avr/io.h>         // Accès bas niveau au microcontrôleur

#include <util/delay.h>     // Fonctions de temporisation

#include <Wire.h>           // Communication I2C

#include <VL53L1X.h>        // Bibliothèque du capteur de distance

// ---------- Broches XSHUT ----------

// XSHUT permet d'allumer / éteindre un capteur LiDAR

const int xshut1 = 11;      // Capteur 1

const int xshut2 = 12;      // Capteur 2

// ---------- Déclaration des capteurs ----------

VL53L1X lidar1;             // Premier capteur

VL53L1X lidar2;             // Deuxième capteur

int main()

{

   init(); // Initialise Arduino

   // INITIALISATION COMMUNICATION PC (SERIAL)

   Serial.begin(115200);    // Communication avec le PC
```

```
// INITIALISATION BUS I2C
Wire.begin();           // Démarre le bus I2C
Wire.setClock(400000);  // Vitesse rapide (400 kHz)
// CONFIGURATION DES BROCHES XSHUT
pinMode(xshut1, OUTPUT);   // Broche capteur 1 en sortie
pinMode(xshut2, OUTPUT);   // Broche capteur 2 en sortie
// On coupe les deux capteurs au démarrage
digitalWrite(xshut1, LOW);
digitalWrite(xshut2, LOW);
delay(100); // Temps pour bien tout réinitialiser
// INITIALISATION CAPTEUR 1
digitalWrite(xshut1, HIGH); // On active seulement capteur 1
delay(100);                 // Temps de démarrage capteur
lidar1.setTimeout(500);     // Si pas de réponse -> erreur
if (!lidar1.init())         // Test si capteur détecté
{
    Serial.println("Erreur lidar 1");
    while (1);              // Bloque le programme si erreur
}
lidar1.setAddress(0x2A);    // On change son adresse I2C
lidar1.setDistanceMode(VL53L1X::Long); // Mode longue distance
lidar1.startContinuous(50);  // Mesure automatique toutes les 50 ms
// INITIALISATION CAPTEUR 2
```

```
digitalWrite(xshut2, HIGH);  // On active capteur 2
delay(100);                  // Temps de démarrage
lidar2.setTimeout(500);      // Timeout de sécurité
if (!lidar2.init())          // Test capteur 2
{
    Serial.println("Erreur lidar 2");
    while (1);               // Stop si erreur
}
// Le capteur 2 garde l'adresse par défaut 0x29
lidar2.setDistanceMode(VL53L1X::Long); // Même mode
lidar2.startContinuous(50);  // Mesure automatique
Serial.println("Deux lidars prêts");
// BOUCLE PRINCIPALE
while (1)
{
    // Lecture distance capteur 1 (en mm)
    int distance1 = lidar1.read();
    // Lecture distance capteur 2 (en mm)
    int distance2 = lidar2.read();
    // Affichage des distances sur le moniteur série
    Serial.print("Lidar 1 : ");
    Serial.println(distance1);
    Serial.print(" mm   ");
```

```
        Serial.println("Lidar 2 : ");
        Serial.print(distance2);
        Serial.println(" mm");
        delay(100); // Petite pause pour éviter surcharge
    }
}
```

Suivi de mur avec correcteur proportionnel

Le suivi de mur repose sur un correcteur proportionnel. L'objectif est de maintenir une distance constante entre le robot et le mur afin d'assurer une trajectoire stable.

L'erreur est calculée comme la différence entre la distance souhaitée et la distance mesurée. Cette erreur est ensuite multipliée par un gain proportionnel Kp afin de corriger la trajectoire.

Lorsque le robot est trop proche du mur, il ralentit un côté pour s'éloigner. À l'inverse, s'il est trop loin, il corrige dans l'autre direction.

```
float Kp = 0.5;

int consigne = 200;

int mesure = lidar2.read();

int erreur = consigne - mesure;

float action = Kp * erreur;

int moteurGauche = 100 + action;

int moteurDroit = 100 - action;
```

Programme global du robot

Le fonctionnement global du robot est organisé sous forme de machine d'états. Cette organisation permet de gérer les différentes phases du parcours de manière claire et structurée.Le robot peut être en attente de démarrage, en suivi de ligne, en suivi de mur ou en arrêt sur détection d'une marque noire.

```
#include <Wire.h>
#include <VL53L1X.h>
#include <PololuLedStrip.h>
PololuLedStrip<A0> rubanLed;
#define nbLeds 8
rgb_color tableauCouleurs[nbLeds]; // tableau des couleurs des LEDs
VL53L1X lidar1;
VL53L1X lidar2;
// Réglages
int Kp = 3; // Gain proportionnel
int consigne = 200; // Distance cible (en mm)
int vitesseBase = 100; // Vitesse de base des moteurs
int Vmax = 150;
int Vmin = 0;
int seuilNoir = 900; // seuil détection sol noir (capteurs analogiques)
int erreur, action, mesure1,mesure2,moteurGauche, moteurDroit;
const int XSHUT1 = PB3; // broches XSHUT des LIDAR
const int XSHUT2 = PB4; // broches XSHUT des LIDAR
enum state {
 etapeInit, // état initial (attente)
 etapeSuiviDistance, // suivi de distance avec régulation
 etapeArretNoir // arrêt si ligne noire détectée
};
state etapeActive = etapeInit;
state etapeSuivante = etapeInit;
void ADC_init()
{
 // Référence AVcc
 ADMUX = (1 << REFS0); // référence tension = AVcc (5V)
 // Activation ADC + prescaler 128
```

```
 ADCSRA = (1 << ADEN) |(1 << ADPS2) |(1 << ADPS1) |(1 << ADPS0);
}
uint16_t lireADC(uint8_t canal) // lecture d'un canal analogique (A0-A7)
{
 ADMUX = (ADMUX & 0xF0) | (canal & 0x0F); // sélection canal
 ADCSRA |= (1 << ADSC); // démarrage conversion
 while (ADCSRA & (1 << ADSC)); // attente fin conversion
 return ADC; // valeur 0-1023
}
void initFonctionsTempsArduino()
{
 sei(); // activation interruptions globales
 TCCR0A = (1 << WGM01) | (1 << WGM00); // configuration Timer0 en PWM
rapide
 TCCR0B = (1 << CS01) | (1 << CS00);
 TIMSK0 = (1 << TOIE0); // interruption overflow
}
void setVitesse(int gauche, int droite)
{
 if (gauche > Vmax) gauche = Vmax; // saturation des vitesses
 if (gauche < Vmin) gauche = Vmin;
 if (droite > Vmax) droite = Vmax;
 if (droite < Vmin) droite = Vmin;
 OCR1B = gauche; // écriture PWM moteurs (Timer1)
 OCR1A = droite;
}
int main()
{
 init();  // init Arduino bas niveau
 ADC_init(); // init convertisseur analogique
 initFonctionsTempsArduino();
 sei(); // activation interruptions
 // I2C + capteur (bloc 1)
 Wire.begin(); // Démarrage bus I2C
 Wire.setClock(400000); // I2C rapide (400 kHz)
 Serial.begin(115200); // debug
 DDRB |= (1 << XSHUT1); // activation broches XSHUT LIDAR
 DDRB |= (1 << XSHUT2); // activation broches XSHUT LIDAR
 PORTB &= ~(1 << XSHUT1); // on reset les capteurs au départ
 PORTB &= ~(1 << XSHUT2); // on reset les capteurs au départ
```

```
 delay(10);
 PORTB |= (1 << XSHUT1);  // activation LIDAR 1
 lidar1.setTimeout(500); // Timeout lecture capteur (ms)
 if (!lidar1.init()) // Vérification du capteur
 {
   Serial.println("Failed to detect and initialize sensor!");
   while (1); // Blocage si capteur absent
 }
 lidar1.setAddress(0x2A); // changement adresse I2C
 lidar1.setDistanceMode(VL53L1X::Long); // Mode longue portée
 lidar1.setMeasurementTimingBudget(50000); // Temps de mesure (µs)
 lidar1.startContinuous(50); // Mesure continue toutes les 50 ms
 PORTB |= (1 << XSHUT2); // activation LIDAR 2
 lidar2.setTimeout(500);
 if (!lidar2.init())
 {
   Serial.println("Erreur lidar 2");
   while (1);
 }
 lidar2.setDistanceMode(VL53L1X::Long);
 lidar2.setMeasurementTimingBudget(50000);
 lidar2.startContinuous(50);
 // moteurs
 DDRD |= (1 << PD2) | (1 << PD3) | (1 << PD7); // Direction moteurs
 DDRB |= (1 << PB0) | (1 << PB1) | (1 << PB2) | (1 << PB5) ; // PWM
moteurs et bouton
 DDRC &= ~(1 << PC0); // entrée analogique
 DDRC |= (1 << PC0);
 TCCR1A = (1 << COM1A1) | (1 << COM1B1) | (1 << WGM10); // Configuration
PWM Timer1
 TCCR1B = (1 << WGM12) | (1 << CS11);
 PORTD |=  (1 << PD2); // sens moteur gauche
 PORTD &= ~(1 << PD3); // sens inverse désactivé
 PORTD |=  (1 << PD7); // sens moteur droit
 PORTB &= ~(1 << PB0); // sens inverse désactivé;
 PORTB |= (1 << PB5);  //pull-up activée
 while (1)
 {
   bool surNoir = (lireADC(6) > seuilNoir || lireADC(7) > seuilNoir); //
détection ligne noire (capteurs analogiques)
```

```
switch (etapeActive)
  {
    case etapeInit:
  {
  //BATTERIE
  float batt = lireADC(3) * (5.07 / 1023);
  if (batt < 3.0)
  {
    for (int i = 0; i < nbLeds; i++)
        tableauCouleurs[i] = rgb_color(10, 0, 0); // rouge
    etapeSuivante = etapeInit; // blocage
    setVitesse(0, 0);
    break;
  }
  else if (batt < 3.1)
  {
    for (int i = 0; i < nbLeds; i++)
        tableauCouleurs[i] = rgb_color(10, 3, 0); // orange
  }
  else
  {
    for (int i = 0; i < nbLeds; i++)
        tableauCouleurs[i] = rgb_color(0, 10, 0); // vert
  }
  rubanLed.write(tableauCouleurs, nbLeds);
  //BOUTON
  if (!(PINB & (1 << PB5)))
  {
    etapeActive = etapeSuivante;
    etapeSuivante = etapeSuiviDistance;
  }
  else
  {
    setVitesse(0, 0);
    etapeSuivante = etapeInit;
  }
  break;
  }
  case etapeSuiviDistance:
    if (surNoir) {
```

```
          etapeActive = etapeSuivante;
          etapeSuivante = etapeArretNoir;
        }
        else {
           mesure2 = lidar2.read();  // lecture distance LIDAR
           erreur = consigne - mesure2;  // calcul erreur de distance
           action = (Kp * erreur) / 10; // correcteur proportionnel
           moteurGauche = vitesseBase - action; // calcul vitesses moteurs
différentielles
           moteurDroit  = vitesseBase + action; // calcul vitesses moteurs
différentielles
          if (action > Vmax)  // saturation
            {
            action = Vmax;
            }
            if (action < Vmin)
            {
              action = Vmin;
            }
          setVitesse(moteurGauche, moteurDroit);
        }
        break;
      case etapeArretNoir:
        setVitesse(0, 0);
        break;
     }
    }
    // UPDATE ETAT
    etapeActive = etapeSuivante; // mise à jour état automate
    // Serial.print("A6=");
    // Serial.print(capteurMilieuG);
    // Serial.print(" A7=");
    // Serial.print(capteurMilieuD);
    //Debug (optionnel)
    //Serial.print("mesure: ");
    //Serial.println(mesure2);
    // Serial.print(" erreur: ");
    // Serial.println(erreur);
    // Serial.print(sensor.read());
    // if (sensor.timeoutOccurred()) { Serial.print(" TIMEOUT"); }
```

```
   // Serial.println();
}
```

Conclusion

Ce projet a permis de concevoir un robot autonome complet en suivant une démarche progressive allant de l'étude des capteurs jusqu'à la réalisation finale du système. Chaque étape a permis de valider une partie du fonctionnement global, que ce soit la détection de ligne, la conception électronique, la fabrication de la carte ou encore la programmation.

Le robot final est capable de suivre une trajectoire, de détecter des événements sur le parcours et de s'adapter à son environnement grâce aux différents capteurs intégrés.

```
#include <Wire.h>

#include <VL53L1X.h>

#include <PololuLedStrip.h>

PololuLedStrip<A0> rubanLed;

#define nbLeds 8

rgb_color tableauCouleurs[nbLeds]; // tableau des couleurs des LEDs

VL53L1X lidar1;

VL53L1X lidar2;

// Réglages

int Kp = 3; // Gain proportionnel

int consigne = 250; // Distance cible (en mm)

int vitesseBase = 75;

; // Vitesse de base des moteurs

int Vmax = 150;
```

```
int Vmin = 0;

int seuilNoir = 800;

int seuilNoir1 = 800; // seuil détection sol noir (capteurs analogiques)

int correction = 0;

int erreur, action, mesure1,mesure2,moteurGauche, moteurDroit, capteurG
,capteurD;

bool noirD , noirG;

unsigned long tempsChargA1;

unsigned long tempsChargA2;

int compteurA1 = 0;

int compteurA2 = 0;

uint16_t valeurPreceA1 = 0;

uint16_t valeurActuA1 = 0;

uint16_t valeurPreceA2 = 0;

uint16_t valeurActuA2 = 0;

const int XSHUT1 = PB3; // broches XSHUT des LIDAR

const int XSHUT2 = PB4; // broches XSHUT des LIDAR

enum state {

 etapeInit, // état initial (attente)

 etapeSuiviLigne, // suivi de distance avec régulation

 etapeSuiviMur, // arrêt si ligne noire détectée

 etapeRotationDroite,

 etapeRotationGauche,

 etapeArretDefinitif
```

```
};
state etapeActive = etapeInit;
state etapeSuivante = etapeInit;
void setVitesse(int gauche, int droite)
{
     moteurGauche = vitesseBase - action ;
     moteurDroit  = vitesseBase + action ;
 // moteur gauche
 if (gauche >= 0)
 {
   PORTD |=  (1 << PD7);
   PORTB &= ~(1 << PB0);
 }
 else
 {
   PORTD &= ~(1 << PD7);
   PORTB |=  (1 << PB0);
   gauche = -gauche;
 }
 // moteur droit
 if (droite >= 0)
 {
   PORTD |=  (1 << PD2);
```

```
    PORTD &= ~(1 << PD3);
  }
  else
  {
    PORTD &= ~(1 << PD2);
    PORTD |=  (1 << PD3);
    droite = -droite;
  }
  OCR1B = gauche;
  OCR1A = droite;
}
void Marque()
{
  DDRC |= (1 << PC1);   // OUTPUT
  PORTC |= (1 << PC1);  // HIGH
  delayMicroseconds(50);
  DDRC &= ~(1 << PC1);  // INPUT
  unsigned long debutA1 = micros();
   while ((PINC & (1 << PINC1)) &&
          (micros() - debutA1 < 30000))
   {
   }
  tempsChargA1 = micros() - debutA1;
```

```
 // Capteur A2
 DDRC |= (1 << PC2);   // OUTPUT
 PORTC |= (1 << PC2);  // HIGH
 delayMicroseconds(50);
 DDRC &= ~(1 << PC2);  // INPUT
 unsigned long debutA2 = micros();
 while ((PINC & (1 << PINC2)) &&
        (micros() - debutA2 < 30000))
 {
 }
 tempsChargA2 = micros() - debutA2;
}
void ADC_init() //Fonction d'initialisation du convertisseur analogique.
{
 ADMUX = (1 << REFS0); // référence tension = AVcc (5V)
 ADCSRA = (1 << ADEN) |(1 << ADPS2) |(1 << ADPS1) |(1 << ADPS0);//
Activation ADC + prescaler 128
}
uint16_t lireADC(uint8_t canal) // lecture d'un canal analogique (A0-A7)
{
 ADMUX = (ADMUX & 0xF0) | (canal & 0x0F); // sélection canal
 ADCSRA |= (1 << ADSC); // démarrage conversion
 while (ADCSRA & (1 << ADSC)); // attente fin conversion
 return ADC; // valeur 0-1023
```

```
}
int main()
{
 init();  // init Arduino bas niveau
 ADC_init(); // init convertisseur analogique
 sei(); // activation interruptions
 // I2C + capteur (bloc 1)
 Wire.begin(); // Démarrage bus I2C
 Wire.setClock(400000); // I2C rapide (400 kHz)
 Serial.begin(115200); // debug
 DDRB |= (1 << XSHUT1); // activation broches XSHUT LIDAR
 DDRB |= (1 << XSHUT2); // activation broches XSHUT LIDAR
 PORTB &= ~(1 << XSHUT1); // on reset les capteurs au départ
 PORTB &= ~(1 << XSHUT2); // on reset les capteurs au départ
 _delay_ms(20);
 PORTB |= (1 << XSHUT1);  // activation LIDAR 1
 lidar1.setTimeout(500); // Timeout lecture capteur (ms)
 if (!lidar1.init()) // Vérification du capteur
 {
   Serial.println("Failed to detect and initialize sensor!");
   while (1); // Blocage si capteur absent
 }
 lidar1.setAddress(0x2A); // changement adresse I2C
```

```
 lidar1.setDistanceMode(VL53L1X::Long); // Mode longue portée
 lidar1.setMeasurementTimingBudget(50000); // Temps de mesure (µs)
 lidar1.startContinuous(50); // Mesure continue toutes les 50 ms
 PORTB |= (1 << XSHUT2); // activation LIDAR 2
 lidar2.setTimeout(500);
 if (!lidar2.init())
 {
   Serial.println("Erreur lidar 2");
   while (1);
 }
 lidar2.setDistanceMode(VL53L1X::Long);
 lidar2.setMeasurementTimingBudget(50000);
 lidar2.startContinuous(50);
 // moteurs
 DDRD |= (1 << PD2) | (1 << PD3) | (1 << PD7); // Direction moteurs
 DDRB |= (1 << PB0) | (1 << PB1) | (1 << PB2) | (1 << PB5) ; // PWM
moteurs et bouton
 DDRC &= ~(1 << PC0); // entrée analogique
 TCCR1A = (1 << COM1A1) | (1 << COM1B1) | (1 << WGM10); // Configuration
PWM Timer1
 TCCR1B = (1 << WGM12) | (1 << CS11);
 PORTB |= (1 << PB5);  //pull-up activée
 //Serial.println("start");
 while (1)
```

```
 {
   bool surNoir = (lireADC(6) > seuilNoir || lireADC(7) > seuilNoir); //
détection ligne noire (capteurs analogiques) et surNoir = true
   float batt = lireADC(3) * (5.07 / 1023); //Convertit tension analogique
en volts.
   capteurG = lireADC(6);
   capteurD = lireADC(7);
   noirG = (capteurG > seuilNoir);
   noirD = (capteurD > seuilNoir);
   // mise à jour A1
   Marque();
   valeurPreceA1 = valeurActuA1;
   valeurActuA1 = tempsChargA1;
   // mise à jour A2
   valeurPreceA2 = valeurActuA2;
   valeurActuA2 = tempsChargA2;
   //Serial.println(valeurActuA1);
   //Serial.println(valeurActuA2);
   // comptage front montant A1
   if (valeurActuA1 > seuilNoir1  && valeurPreceA1 <= seuilNoir1 )
   {
      compteurA1++;
      //Serial.println(compteurA1);
   }
```

```
// comptage front montant A2
if (valeurActuA2 > seuilNoir1  && valeurPreceA2 <= seuilNoir1 )
{
   compteurA2++;
}

switch (etapeActive)
  {
  case etapeInit:
  Serial.println(batt);
  if (batt < 3.1)
  {
    for (int i = 0; i < nbLeds; i++)
    tableauCouleurs[i] = rgb_color(10, 0, 0); // rouge
    //Serial.println("tension faible");
    etapeSuivante = etapeInit; // blocage
    setVitesse(0, 0);
    rubanLed.write(tableauCouleurs, nbLeds);
    break;
  }
  else if (batt < 3.2)
  {
    for (int i = 0; i < nbLeds; i++)
```

```
      tableauCouleurs[i] = rgb_color(10, 3, 0); // orange
}
else
{
  for (int i = 0; i < nbLeds; i++)
      tableauCouleurs[i] = rgb_color(0, 10, 0); // vert
}
rubanLed.write(tableauCouleurs, nbLeds); //Met à jour le ruban LED.
//BOUTON

if (!(PINB & (1 << PB5))) //Bouton appuyé ?
{
  //Serial.println("appui bp");
  etapeSuivante = etapeSuiviLigne;
}
else
{
  //Serial.println("pas de bp");
  setVitesse(0, 0);
  etapeSuivante = etapeInit;
}
break;
```

```
 case etapeSuiviLigne:
 //Serial.print(etapeSuiviLigne);
// Si plus de ligne noire
// -> passage suivi mur
for(int i=0; i<nbLeds; i++)
    tableauCouleurs[i] = rgb_color(0, 10, 0); // Vert
    erreur = capteurG - capteurD;
    action = erreur / 30;
    moteurGauche = vitesseBase - action - 15;
    moteurDroit  = vitesseBase + action - 15;
    setVitesse(moteurGauche, moteurDroit);
  if (compteurA1 == 2)
  {
    etapeSuivante = etapeRotationDroite;
    vitesseBase = 60;
  }
  if (compteurA2 == 2)
  {
    vitesseBase = 75;
  }
  if (compteurA2 == 5)
  {
    vitesseBase = 80;
```

```
  etapeSuivante = etapeRotationGauche;
}
if (compteurA2 == 7)
{
  for(int i=0; i<nbLeds; i++)
  tableauCouleurs[i] = rgb_color(10, 10, 10);
  vitesseBase=120;
}
if (compteurA2 == 9)
{
  vitesseBase=80;
}
if (compteurA2 == 10)
{
  etapeSuivante = etapeRotationGauche;
}
if (compteurA1 == 11)
{
  etapeSuivante = etapeRotationDroite;
  vitesseBase=80;
}
if (compteurA1 == 12)
{
```

```
      mesure1 = lidar1.read();
      if (mesure1 < 150)
      {
        etapeSuivante = etapeArretDefinitif;
      }
      }

      if (!noirG && !noirD)
      {
        etapeSuivante = etapeSuiviMur;
      }
    break;



// ETAPE SUIVI MUR
     case etapeSuiviMur:
    // Si ligne retrouvée
    // -> retour suivi ligne
    for(int i=0; i<nbLeds; i++)
        tableauCouleurs[i] = rgb_color(10, 10, 0);
    if (noirG || noirD)
    {
      etapeSuivante = etapeSuiviLigne;
```

```
}
else
{
    mesure2 = lidar2.read();
    erreur = consigne - mesure2;
    action = (Kp * erreur) / 10;
    moteurGauche = vitesseBase - action -10;
    moteurDroit  = vitesseBase + action -10;
    setVitesse(moteurGauche, moteurDroit);
}
break;

case etapeRotationDroite:
for(int i=0; i<nbLeds; i++)
    tableauCouleurs[i] = rgb_color(10, 0, 10);
   setVitesse(200, -200);   // tourne à droite
if (valeurPreceA1 < 20000 && valeurActuA1 >= 20000 )
{
  etapeSuivante = etapeSuiviLigne;
}
  break;

  case etapeRotationGauche:
```

```
      for(int i=0; i<nbLeds; i++)
        tableauCouleurs[i] = rgb_color(0, 10, 10);
       setVitesse(-200,200);   // tourne à droite
    if (valeurPreceA2 < 20000 && valeurActuA2 >= 20000 )
    {
      etapeSuivante = etapeSuiviLigne;
    }
      break;

    case etapeArretDefinitif:
    for(int i = 0; i < nbLeds; i++)
    tableauCouleurs[i] = rgb_color(10, 0, 0); // rouge
    setVitesse(0, 0);
    break;

    }
    rubanLed.write(tableauCouleurs, nbLeds);
    etapeActive = etapeSuivante;// mise à jour état automate
// Serial.print("A1=");
// Serial.print(compteurA1);
// Serial.print(" | A2=");
// Serial.print(compteurA2);
// Serial.println();
```

```
    // Serial.print("A6=");
    // Serial.print(capteurMilieuG);
    // Serial.print(" A7=");
    // Serial.print(capteurMilieuD);
    //Debug (optionnel)
    //Serial.print("A1=");
    //Serial.println(A1);
    //Serial.print(" | A2=");
    //Serial.println(A2);
    //Serial.print("mesure: ");
    //Serial.println(mesure2);
    // Serial.print(" erreur: ");
    // Serial.println(erreur);
    // Serial.print(sensor.read());
    // if (sensor.timeoutOccurred()) { Serial.print(" TIMEOUT"); }
    // Serial.println();
}
}
```

[[RAPPORT FINALE]]

```
#include <Arduino.h>
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

// --- Configuration du Seuil de Ligne ---
const int SEUIL_NOIR = 850;

// --- Fonctions de contrôle des moteurs ---

//Pilote le moteur Gauche
void moteurGauche(int vitesse, bool avance, bool recule) {
   // Broche D7 (PD7) -> Marche avant Gauche
   if (avance)  PORTD |= (1 << PD7);
   else         PORTD &= ~(1 << PD7);

   // Broche D8 (PB0) -> Marche arrière Gauche
   if (recule)  PORTB |= (1 << PB0);
   else         PORTB &= ~(1 << PB0);

   OCR1A = vitesse; // Contrôle la vitesse (Pin D9)
}

//Pilote le moteur Droit (Sens Corrigé)
void moteurDroite(int vitesse, bool avance, bool recule) {
   // Broche D2 (PD2) -> Marche avant Droite
   if (avance)  PORTD |= (1 << PD2);

   // Broche D3 (PD3) -> Marche arrière Droite
   if (recule)  PORTD |= (1 << PD3);
   else         PORTD &= ~(1 << PD3);

   OCR1B = vitesse; // Contrôle la vitesse (Pin D10)
}

// --- Mesure des capteurs externes par temps de décharge (A1 et A2) ---
unsigned long mesurerTempsDecharge(uint8_t pinBit) {
   DDRC |= (1 << pinBit);
   PORTC |= (1 << pinBit);
   delayMicroseconds(50);
```

```
    DDRC &= ~(1 << pinBit);
    unsigned long debut = micros();

    while ((PINC & (1 << pinBit)) && (micros() - debut < 30000)) {
        // Attente active
    }
    return micros() - debut;
}

// --- Programme Principal ---
int main() {
    // 1. DIRECTION DES BROCHES (Sorties moteurs)
    DDRB |= (1 << PB0) | (1 << PB1) | (1 << PB2); // D8, D9 (PWM G), D10
(PWM D)
    DDRD |= (1 << PD7) | (1 << PD3) | (1 << PD2); // D7, D3, D2

    // 2. CONFIGURATION STRICTE DU TIMER 1 À 20 kHz PILE
    TCCR1A = 0;
    TCCR1B = 0;

    // Mode 14 : Fast PWM avec TOP = ICR1
    TCCR1A |= (1 << WGM11) | (1 << COM1A1) | (1 << COM1B1);
    TCCR1B |= (1 << WGM13) | (1 << WGM12);

    // Valeur max (TOP) = 799 avec prédiviseur de 1.
    // Calcul : 16 000 000 Hz / (1 * (799 + 1)) = 20 000 Hz = 20 kHz.
    ICR1 = 799;
    TCCR1B |= (1 << CS10); // Prédiviseur à 1

    // Arrêt initial des moteurs
    OCR1A = 0;
    OCR1B = 0;
    PORTD &= ~((1 << PD7) | (1 << PD3) | (1 << PD2));
    PORTB &= ~(1 << PB0);

    // 3. ARDUINO CORE INITIALISATION
    init();
    DDRC &= ~((1 << PC1) | (1 << PC2)); // A1 et A2 en entrées
```

```
    Serial.begin(115200);
    sei();

    // 4. BOUCLE PRINCIPALE
    while (1) {
        int ligne_g = analogRead(A6);
        int ligne_d = analogRead(A7);

        unsigned long tempsChargA1 = mesurerTempsDecharge(PC1);
        unsigned long tempsChargA2 = mesurerTempsDecharge(PC2);

        // --- Affichage Debug ---
        Serial.print("A6=");   Serial.print(ligne_g);
        Serial.print(" | A7="); Serial.print(ligne_d);
        Serial.print(" | A1="); Serial.print((tempsChargA1 > 100) ? "1" :
"0");
        Serial.print(" | A2="); Serial.println((tempsChargA2 > 100) ? "1" :
"0");

        _delay_ms(10);

        // --- LOGIQUE DE SUIVI DE LIGNE (Fréquence bridée à 20kHz,
vitesses minimales) ---
        int v = 200;
        moteurGauche(((ligne_g/6)+50), true, false);
        moteurDroite(((ligne_d/6)+50), true, false);
    }
}
```

```
void Marque()
{
 DDRC |= (1 << PC1);    // OUTPUT
 PORTC |= (1 << PC1);   // HIGH
 delayMicroseconds(50);
 DDRC &= ~(1 << PC1);   // INPUT
 unsigned long debutA1 = micros();
  while ((PINC & (1 << PINC1)) &&
         (micros() - debutA1 < 30000))
  {
  }
 tempsChargA1 = micros() - debutA1;
 // Capteur A2
 DDRC |= (1 << PC2);    // OUTPUT
 PORTC |= (1 << PC2);   // HIGH
 delayMicroseconds(50);
 DDRC &= ~(1 << PC2);   // INPUT
 unsigned long debutA2 = micros();
 while ((PINC & (1 << PINC2)) &&
         (micros() - debutA2 < 30000))
 {
 }
 tempsChargA2 = micros() - debutA2;
}

   Marque();
   valeurPreceA1 = valeurActuA1;
   valeurActuA1 = tempsChargA1;
   // mise à jour A2
   valeurPreceA2 = valeurActuA2;
   valeurActuA2 = tempsChargA2;
   // comptage front montant A1

unsigned long tempsChargA1;
unsigned long tempsChargA2;
int compteurA1 = 0;
int compteurA2 = 0;
uint16_t valeurPreceA1 = 0;
uint16_t valeurActuA1 = 0;
uint16_t valeurPreceA2 = 0;
```

```
uint16_t valeurActuA2 = 0;
int seuilNoir1 = 800;
```