# SS:SmartShift

# (Changement de vitesse automatique du vélo)

**Nour El Idrissi Abdelaziz**

**N° SCEI :18555**

# Plan de présentation

- **Introduction**
- **Problématique**
- **Cahier des charges**
- **Objectifs et expériences**
- **Conclusion**

# Introduction

# Changement de vitesse du vélo :

4

- il consiste à changer les rapports de transmission entre les disques afin de permettre au cycliste d'adapter sa vitesse avec les conditions de la route.

# Description de système:

- La commande du dérailleur par un moteur
- La commande du moteur selon les données acquises par les capteurs


**Figure 1 : Dérailleur du vélo**

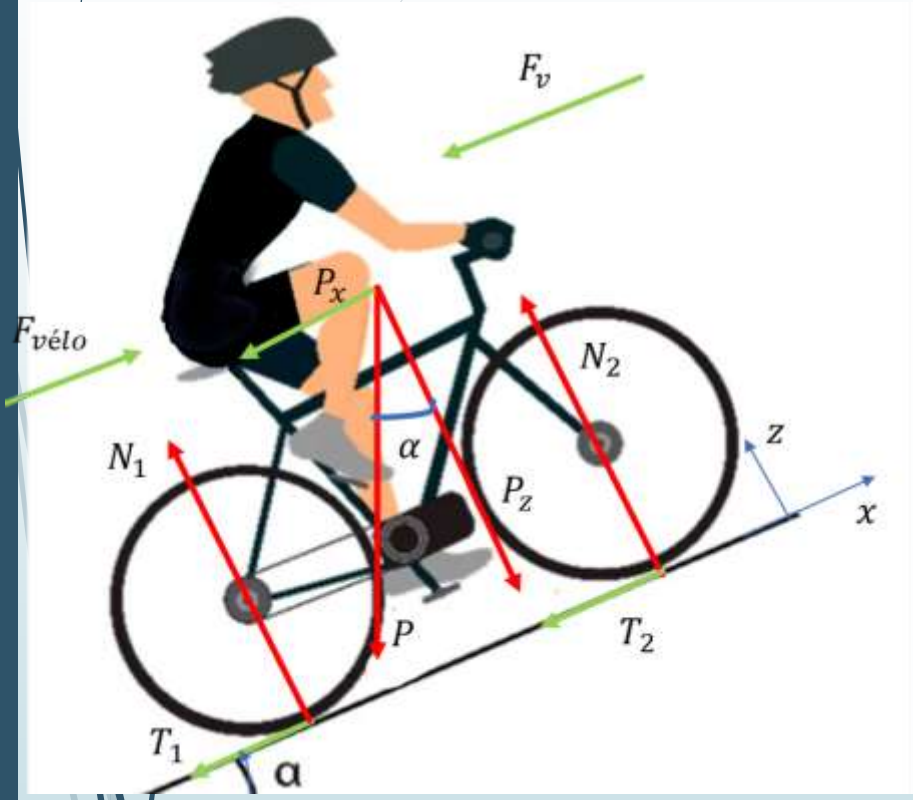



**Figure 2 : Actions mécaniques appliqués sur vélo**

- *En applique le principe fondamental de la dynamique :*
- $m \times \vec{a} = \sum \overrightarrow{force}$
- Projection sur $\overrightarrow{x_1}$ : $m \times a_x = F_{vélo} - F_t$ avec:

$$Ft = m \times g \times \sin(\alpha) + 2 \times f \times N \pm k \times (V_r)^2$$

Et $V_r = v \pm |V_v|$

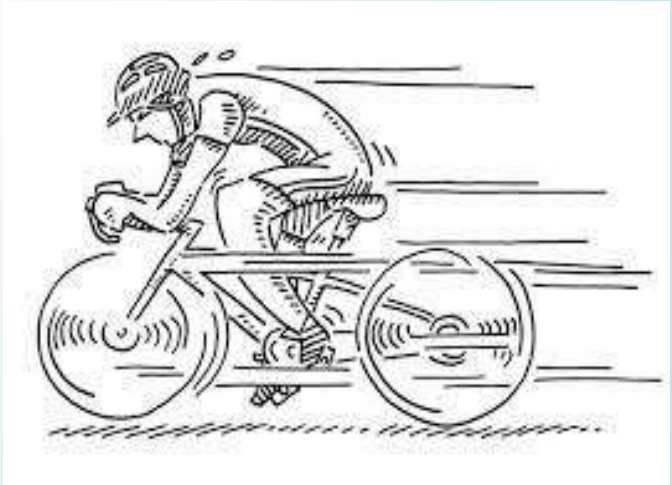
- Projection sur $\overrightarrow{z_1}$ : $m \times a_z = 2 \times N - P_z = 0$ ,donc: $N = \frac{1}{2} \times m \times g \times \cos(\alpha)$
- Or: $\mu = \frac{C_{r \times \omega_r}}{C_p \times \omega_p} = \frac{F_v \times R}{C_p} \times r$
- Alors $r = \frac{C_p \times \mu}{R \times (m \times a_x + m \times g \times \sin(\alpha) + 2 \times f \times N + k \times (V_r)^2)}$

- On suppose qu'on travaille dans un repère galillien
- m : la masse totale (vélo plus cycliste)
- a : accélération de l'ensemble(vélo + cycliste)
- Vr : Vitesse du vélo
- Vv : Vitesse du vent
- μ : le rendement
- Ft : force résistif
- Fvélo : force qui pousse le vélo ( appliquée sur la roue )

# Problématique

- Comment le système peut changer automatiquement les rapports de vitesse du vélo en fonction des facteurs extérieurs ?

- Comment le système peut s'adapter avec la cadence de conduite souhaitée par le cycliste ?



# Solution:

- SS: SmartShift : un changement de vitesse automatique du vélo qui gère les rapports de vitesse automatiquement sans l'intervention du cycliste.

# Cahier des charges

# Cas d'utilisation

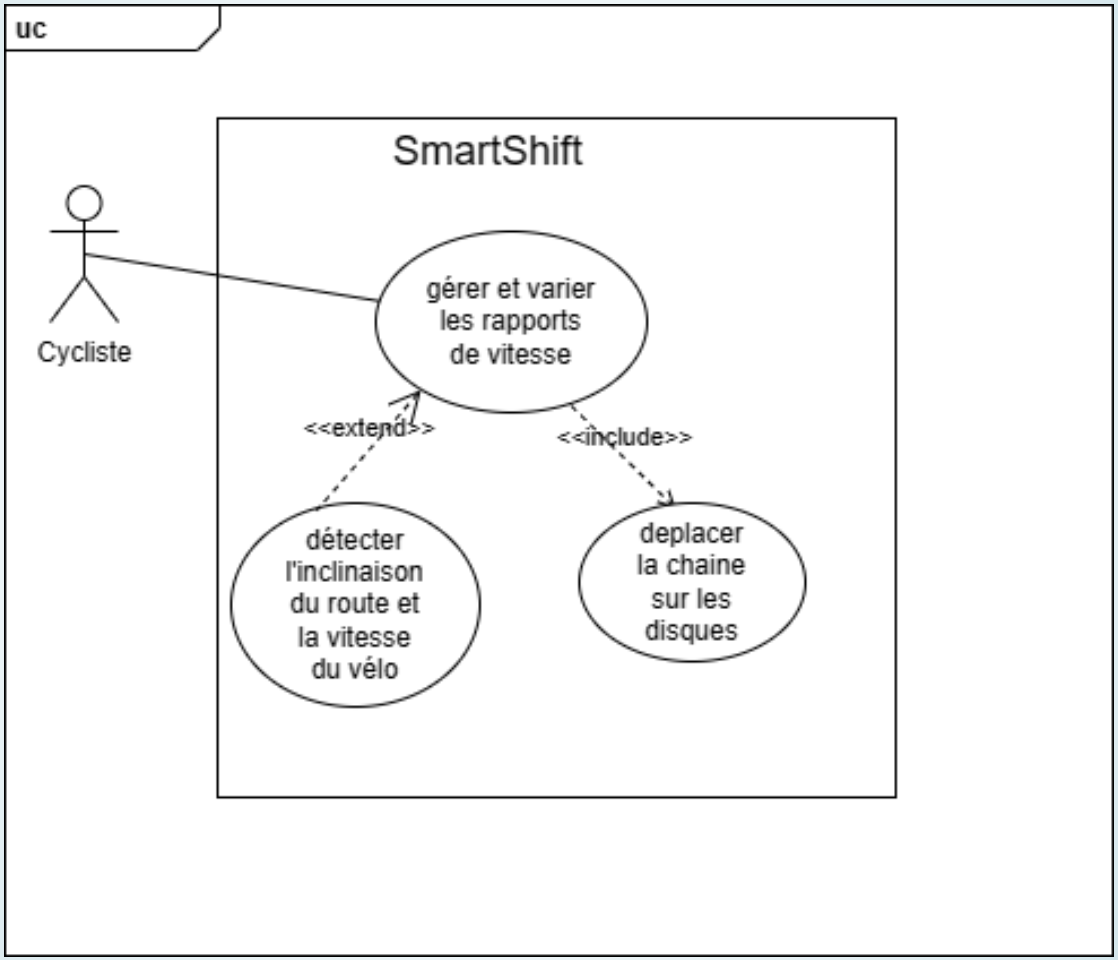



**Figure 3 : Diagramme de cas d'utilisation**

# Diagramme d'exigence

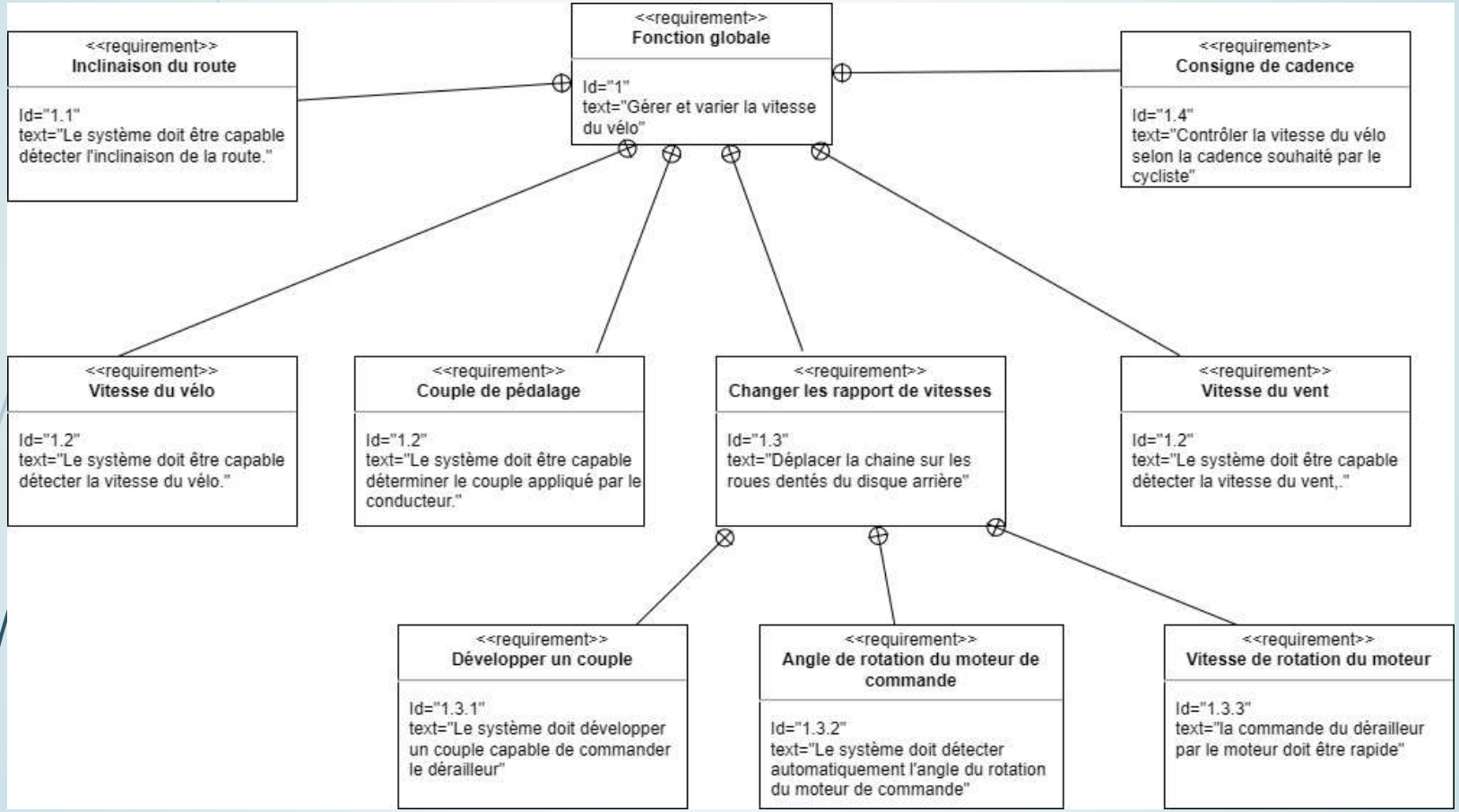



**Figure 4 : Diagramme d'exigences**

# Diagramme de définition de blocs

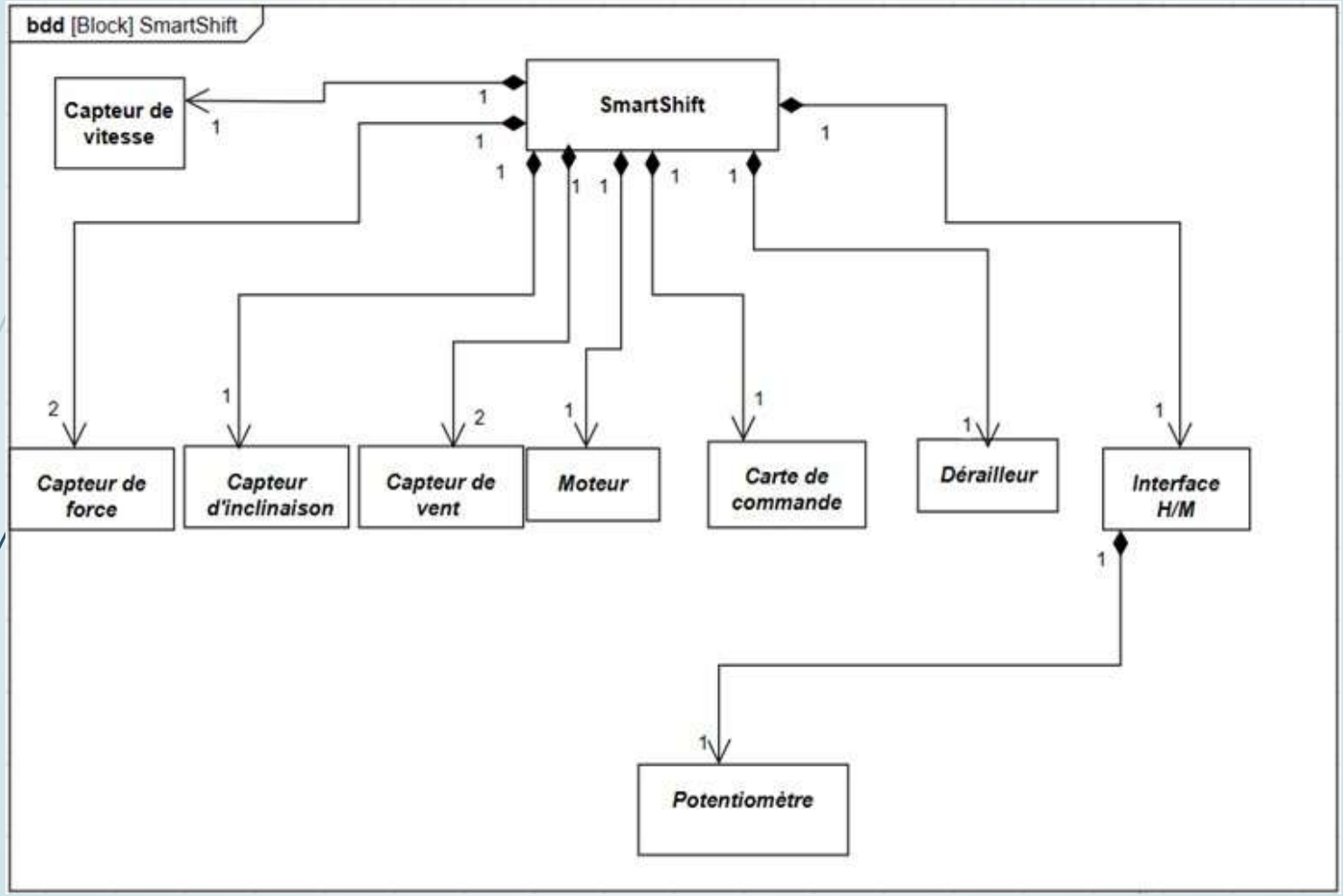



**Figure 5 : Diagramme de définition de blocs**

# Objectifs

# 1er objectif : Proposer et choisir les capteurs convenables à mon système.



# 1-Le chois du capteur de vitesse :

- **Capteur magnétique**

- **Capteur infrarouge**

# Capteur infrarouge

- Avantages:

Détection sans contact - précision - Rapidité

- Incovénients :

Sensiblité aux interferences - dépend des conditions enviramentales

# Capteur magnétique

- Avantages :

Durabilité - Détection sans contact- Grande précision et répétabilité - Facile à intégré

- Incovénients :

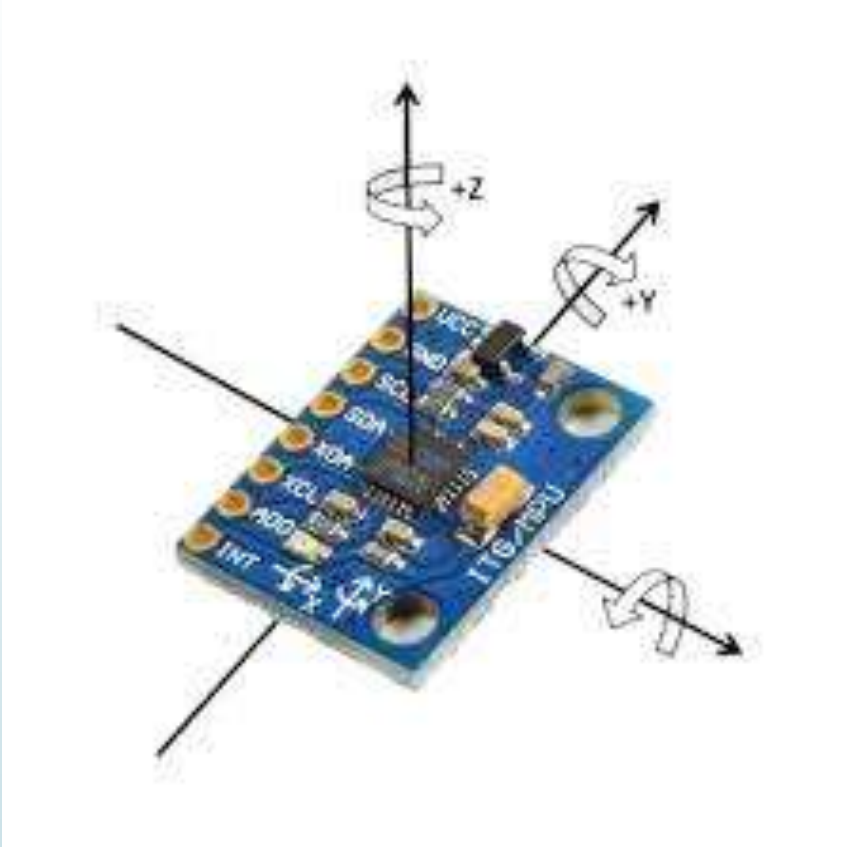
Sensibilité aux champs magnétiques externes - compléxité de calibration

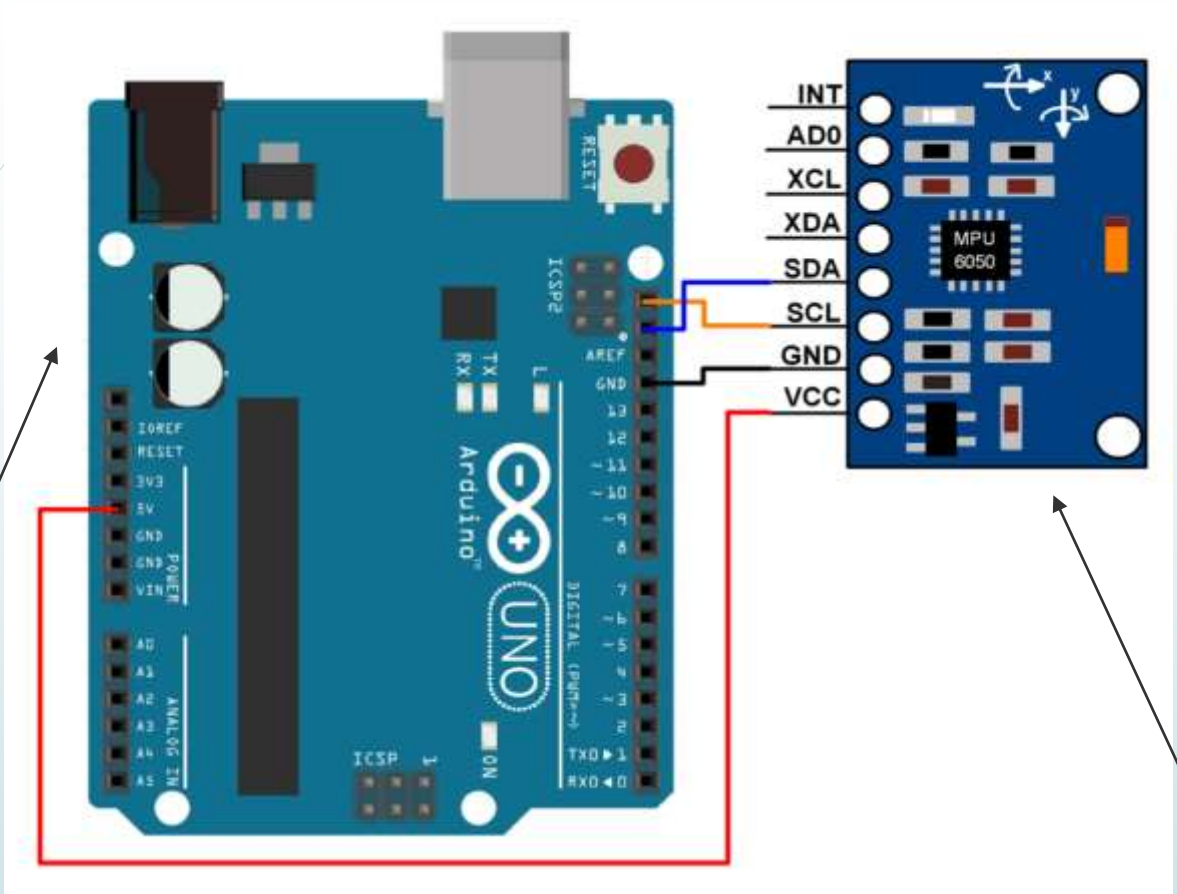
20

# 2-Choix du capteur d'inclinaison

# Gyroscope MPU-6050

**Ce capteur représente un meilleur choix pour notre exigence . En effet il peut mesurer l'inclinaison , la vitesse et aussi l'accélération.**






Arduino UNO

**Figure 6 : Montage du gyroscope MPU-6050**

Gyroscope MPU-6050

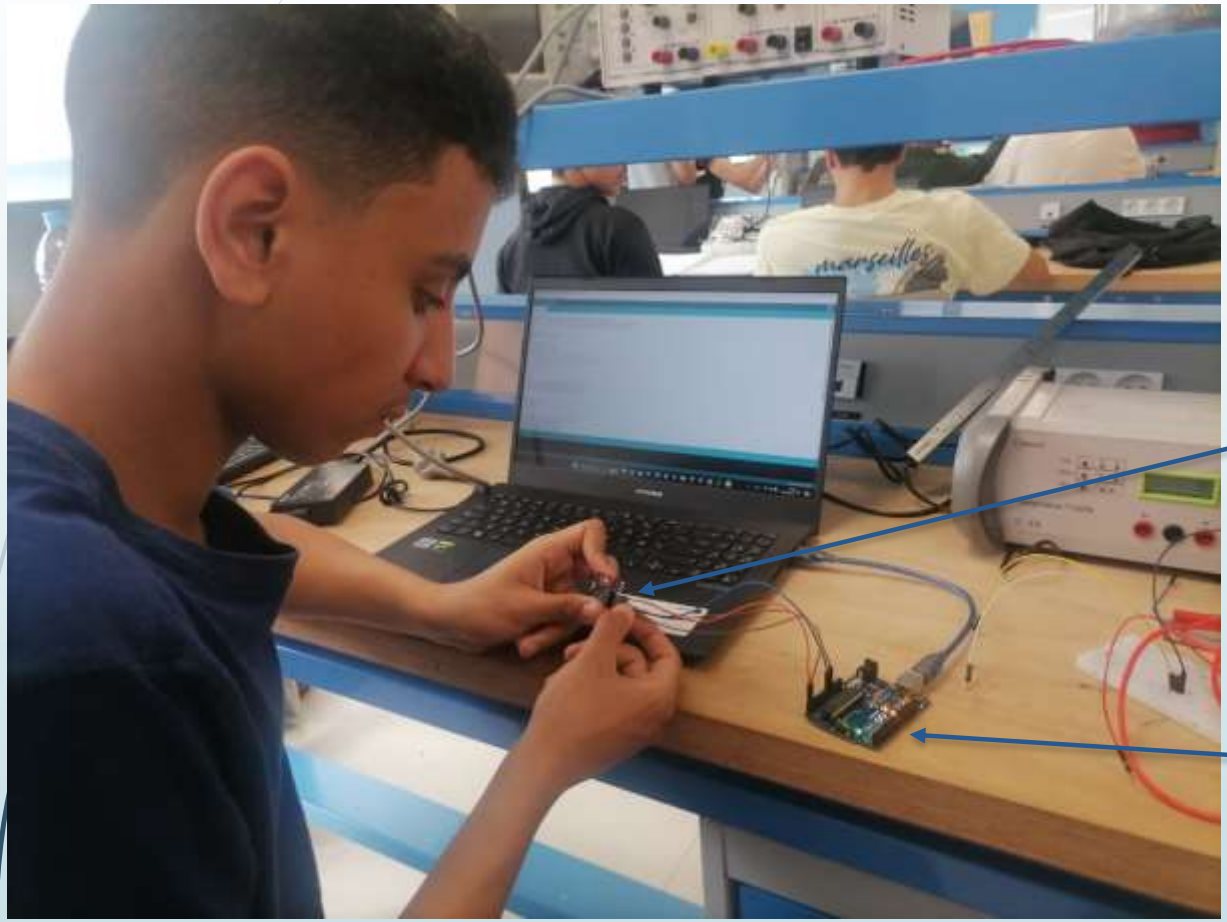



**Figure 7 : Montage réel de gyroscope**

# ESP-32

- On utilise ce microprocesseur pour envoyer les données du capteur à l'arduino


**Figure 8 : Montage réel de l' ESP32 avec gyroscope MPU-6050**

# 2ème objectif : Dimensionner et choisir le moteur à utiliser

# 1-Moteur pas à pas :

- Pour notre système , le moteur pas à pas constitue un excellent choix , puisqu'on veut déplacer la chaine d'un plateau à un autre avec une distance connue entre ces deux derniers


**Figure 9: Moteur pas à pas**

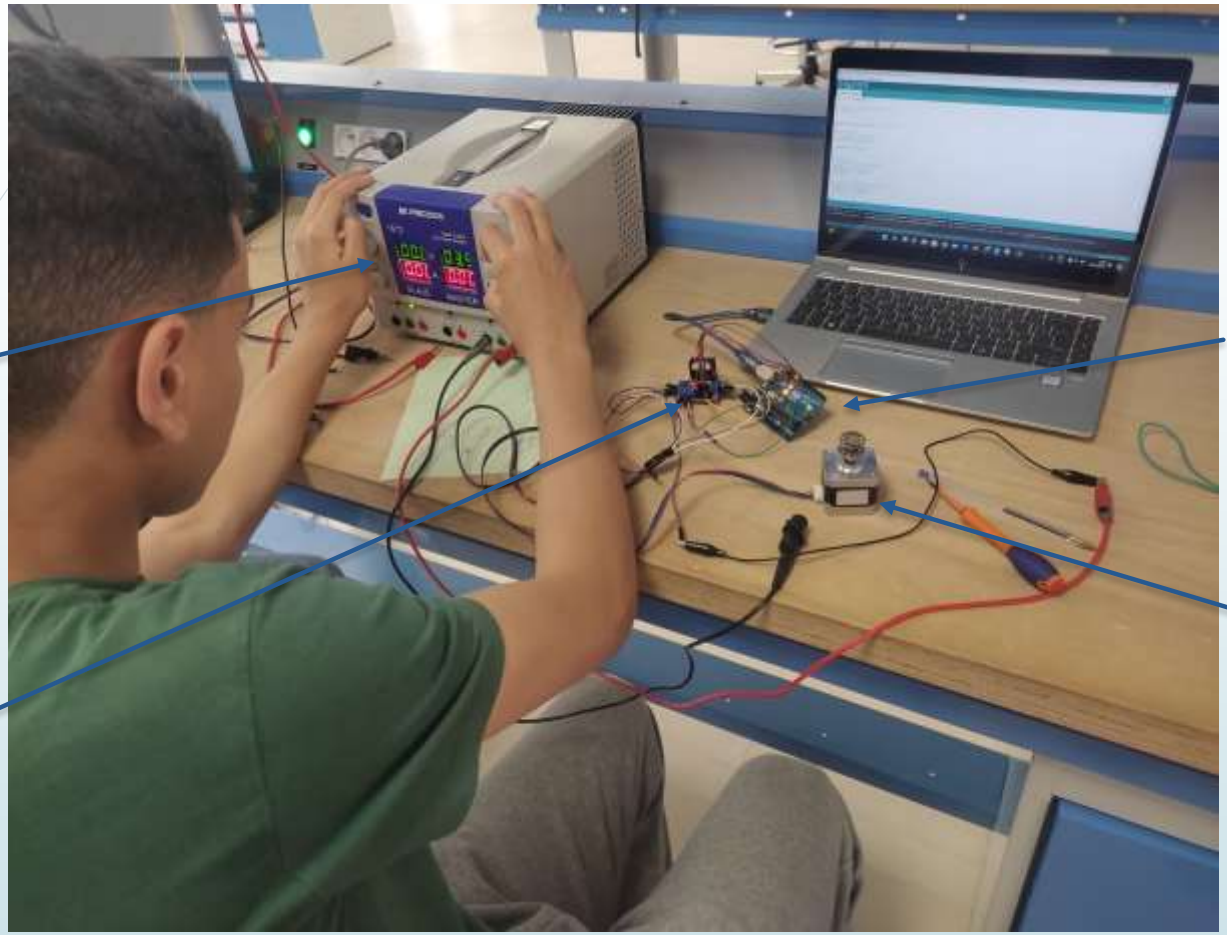



**Figure 10 : montage expérimentale du moteur pas à pas**

- Pour choisir le moteur convenable pour notre système , il faut que son couple soit supérieur au couple appliqué par le dérailleur.

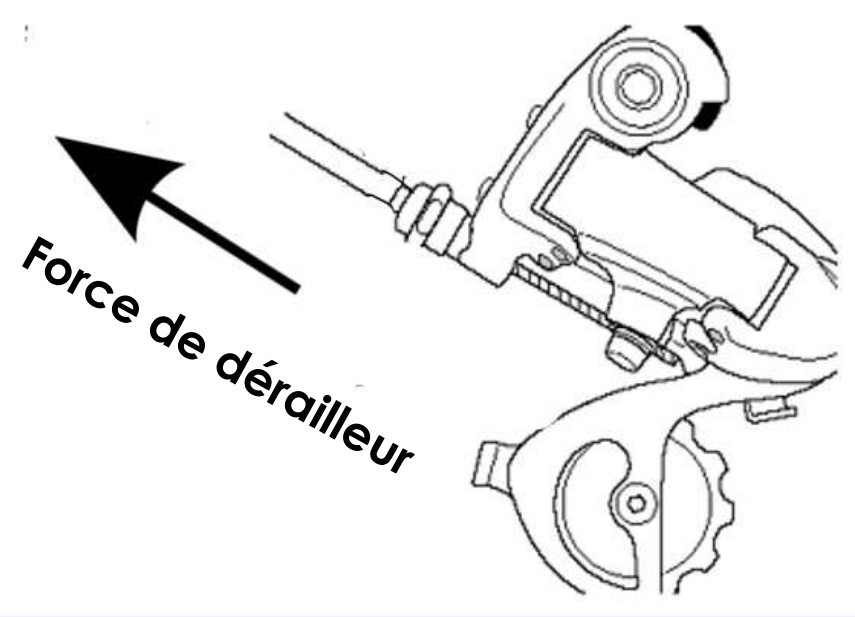



**Figure 11 : Force de tirage du dérailleur**

# Utilisation du capteur d'effort

- On a accroché le capteur d'effort avec le cable du dérailleur
- Effort de dérailleur trouvé : 28,63 N
- Le couple donc est : 0,56 N.m

**Figure 12 : Expérience pour trouver l'effort appliqué par le dérailleur**

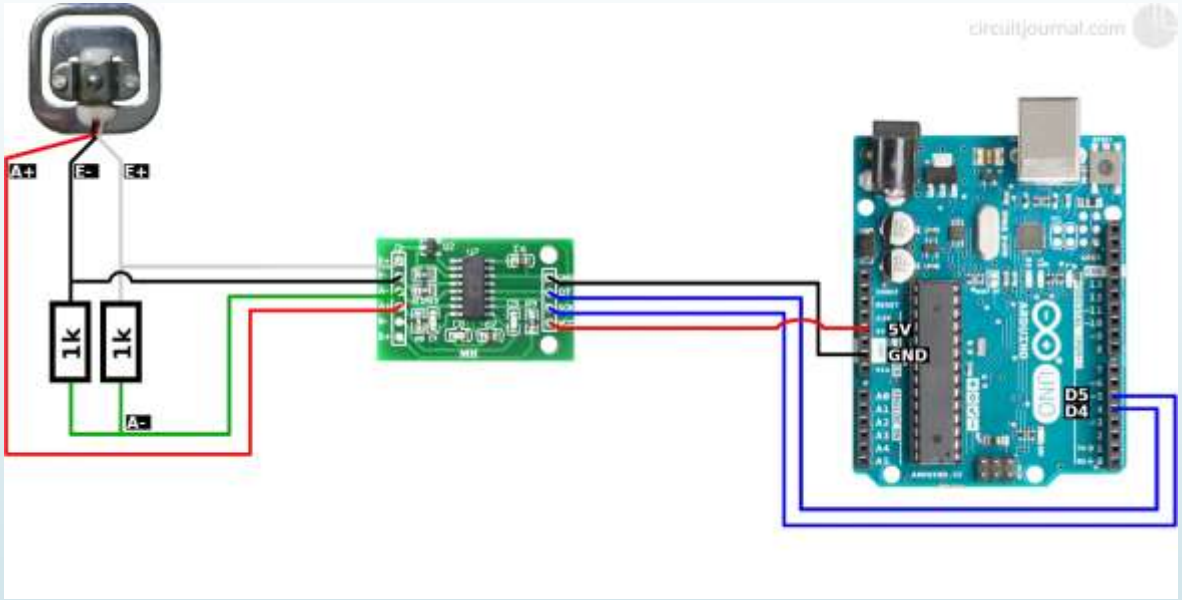
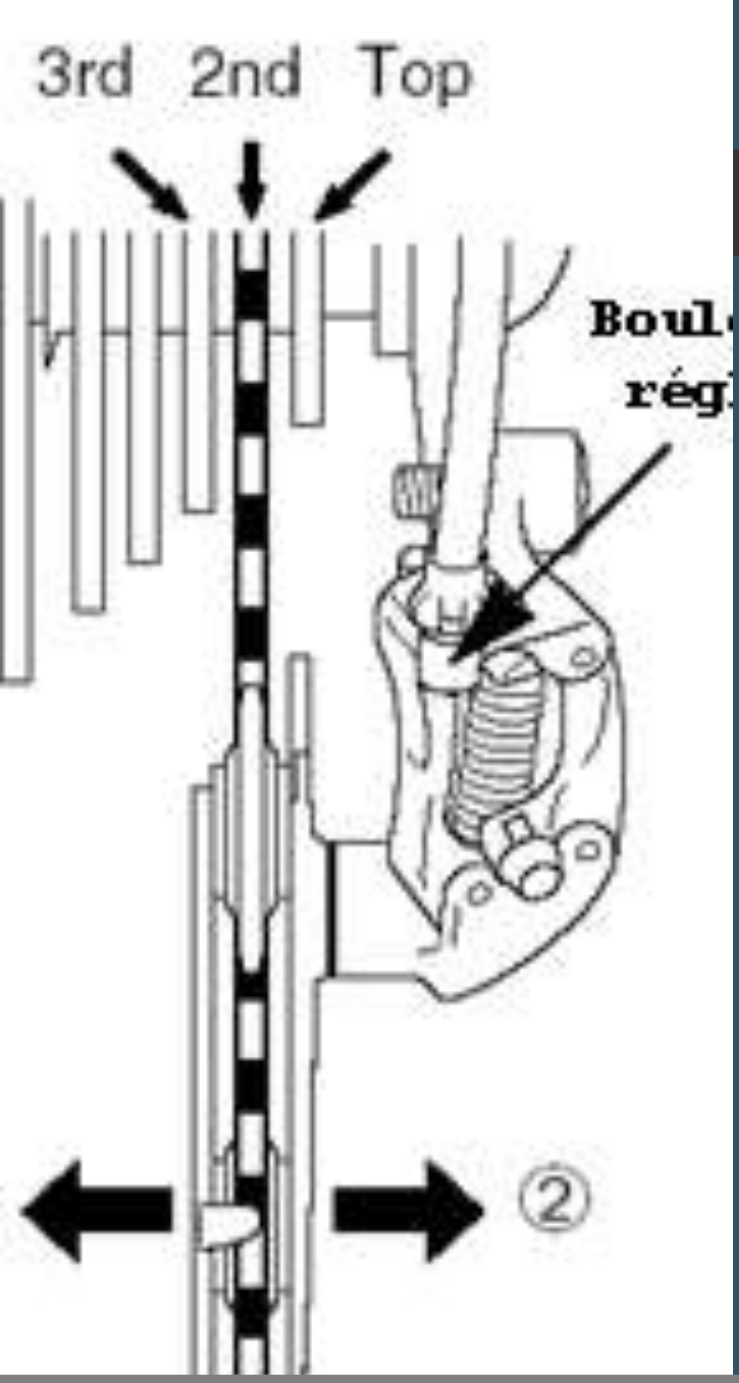



**Figure 13 : Shéma de montage du capteur d'effort avec arduino**






2- Déterminer l'angle de rotaion du moteur associé à la longeur du fil déplacée pour déplacer la chaine d'un plateau à un autre

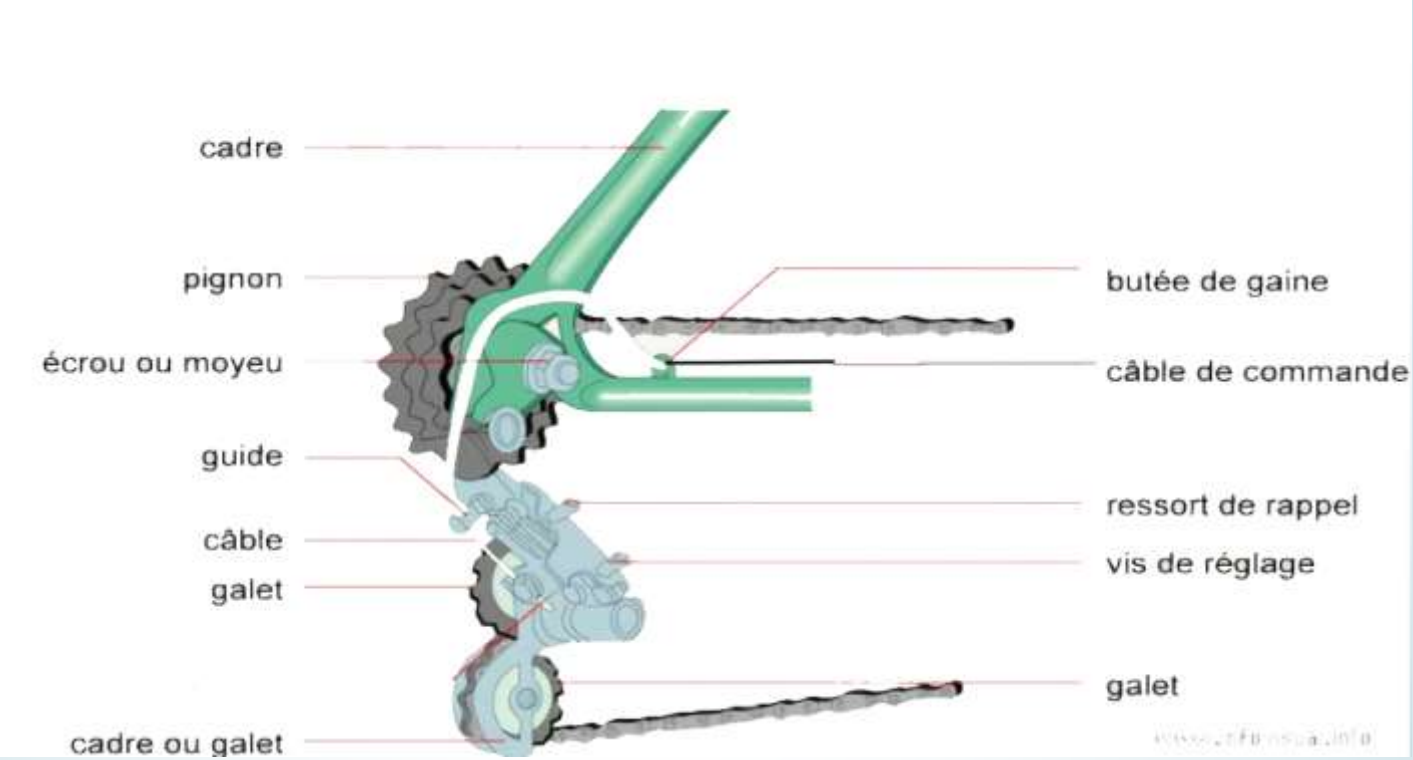



**Figure 14 : Dérailleur de vélo**



Calcule de la longueur du fil nécessaire pour déplacer la chaine d'un pignon a un autre

- En utilisant cette relation : $L = R \times \beta$

Avec :

- L: longueur de fil q'on doit tirer (1 cm trouvée par expérience)
- R : Rayon du tambour sur l'arbre moteur
- β : l'angle que le moteur doit tourner

3ème objectif : Réguler le système en accord avec les préférences de la cadence du cycliste tout en assurant une communication entre le cycliste et le système.

# Régulation du système en accord avec les préférences du cycliste:

- En utilisant un potentiomètre ; le cycliste va spécifier l'accélération qu'il veut atteindre.
- En remplaçant cette accélération dans la relation trouvé . Le système va changer vers le rapport convenable pour atteindre cette accélération

- L'accélération maximale atteinte par les cyclistes professionnels est :1,389 $m/s^2$
- L'accélération du démarrage est :18,75 $m/s^2$

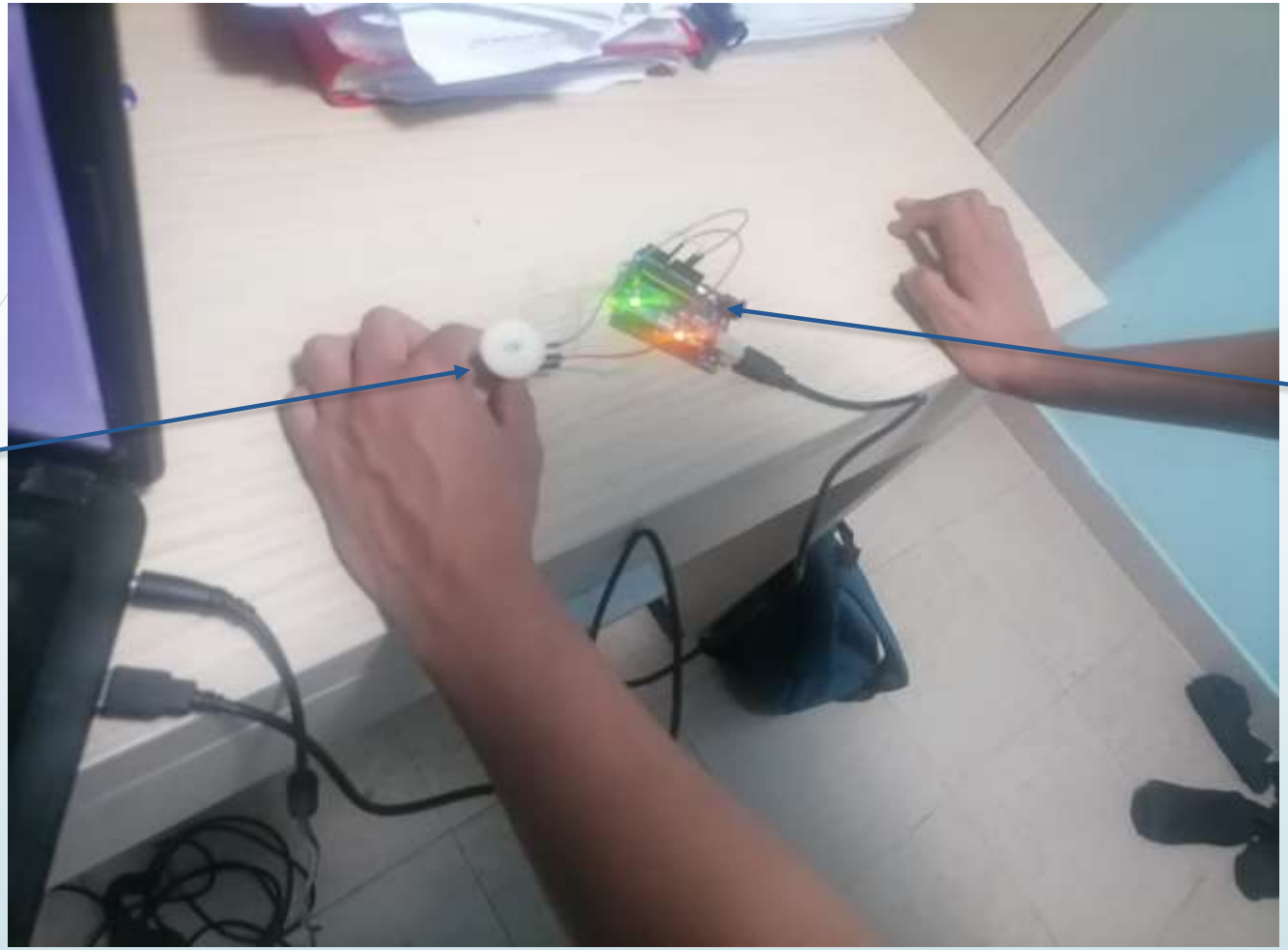



**Figure 15: montage expérimentale du potentiométre avec Arduino**

# Ecran LCD:

- Pour assurer une communication entre le cycliste et le sytème, on a utilisé un écran LCD qui va assurer d'afficher la vitesse du vélo plus le rapport en train d'utilisation

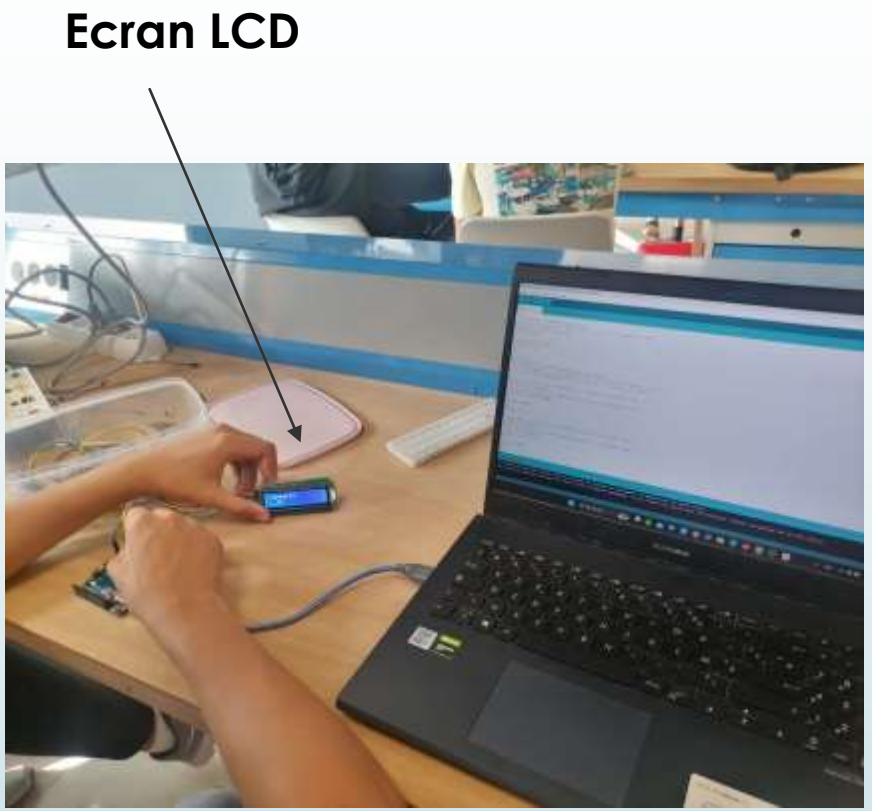



**Figure 16 : montage LCD avec Arduino**

# 4ème objectif : Identifier les paramètres extérieurs qui influencent sur le système.

# 1-Determination du rendement du mécanisme pignon chaine

- Protocole experimentale :
- Lier la roue arrière avec un poids connu , et commencer à appliquer un couple sur la pédale jusqu'à arriver au point ou le poids est en équilibre sans toucher le sol , donc :
- $n = (P*R*r)/Cp$

- n : le rendement
- R : rayon de la roue
- Cp : couple appliqué sur le pédale
- r : le rapport de vitesse

- L'effort trouvé est : 66,28 N

- En faisant l'application numérique , on trouve que le rendement est égale :0,78

# 5ème objectif : Réaliser un prototype qui répond au cahier de charge

# Réalisation du prototype:

- Le capteur de vitesse est dérigé vers la roue
- L'arbre du moteur doit être au même niveau que le fil de dérailleur
- Le capteur d'effort doit être sur la pédale
- La batterie LIPO est à côté d'Arduino pour assurer l'alimentation


BEST
ND FIRST

# Conclusion



# Merci pour votre attention

ANNEXES

# Annexe 1

## Ressources :


- [1] Effigear : la boite à vitesse créer par valeo et effigear : Valeo Cyclee (effigear.com)
- [2] : Enviolo : Information sur la boite à vitesse enviolo et ses composants :https://enviolo.com/wp-content/uploads/Technical-Manual-enviolo-CY2021_June-2021.pdf
- [3] : Bikelec :Description du système de transmission (moyeu à variation continue) de NuVinci :NuVinci - Moyeu à variation continue - Bikelec Blog - Vélos Électriques
- [4] : ebike24 : boutique **en ligne spécialisée dans la vente de vélos électriques et de pièces détachées pour les vélos électriques** : Enviolo - Moyeu à vitesses intégrées Heavy Duty (ebike24.fr)
- [5] : Support enviolo, le support technique et l'assistance pour les produits Enviolo :Spécification des moyeux à vitesses intégrés enviolo Trekking – enviolo
- [6] : ebike24 : boutique **en ligne spécialisée dans la vente de vélos électriques et de pièces détachées pour les vélos électriques** :Enviolo - Kit de conversion Automatic+ vers AutomatiQ (ebike24.fr)
- [7] : ebike24 : boutique **en ligne spécialisée dans la vente de vélos électriques et de pièces détachées pour les vélos électriques** : Enviolo - Commande au guidon Cliq Pro (ebike24.fr)
- [8] : LECYCLO : Présentation des composant et du fonctionnement du dérailleur : Tout savoir sur le dérailleur arrière de vélo (lecyclo.com)

# Annexe 2



## Code Arduino pour MPU –6050:

```
#include "Wire.h"

const int MPU_addr=0x68; int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;

int minVal=265; int maxVal=402;

double x; double y; double z;

void setup(){
    Wire.begin();
    Wire.beginTransmission(MPU_addr);
    Wire.write(0x6B);
    Wire.write(0);
    Wire.endTransmission(true);
    Serial.begin(9600);
}
void loop(){
     Wire.beginTransmission(MPU_addr);
     Wire.write(0x3B);
     Wire.endTransmission(false);
     Wire.requestFrom(MPU_addr,14,true);
     AcX=Wire.read()<<8|Wire.read();
     AcY=Wire.read()<<8|Wire.read();
     AcZ=Wire.read()<<8|Wire.read();
     int xAng = map(AcX,minVal,maxVal,-90,90);
     int yAng = map(AcY,minVal,maxVal,-90,90);
     int zAng = map(AcZ,minVal,maxVal,-90,90);

     x= RAD_TO_DEG * (atan2(-yAng, -zAng)+PI);
     y= RAD_TO_DEG * (atan2(-xAng, -zAng)+PI);
     z= RAD_TO_DEG * (atan2(-yAng, -xAng)+PI);
     if (x < 0) {
   x += 180;
  } else {
    x -= 180;
  }

    x += 180;
  } else {
    x -= 180;
  }

  if (y < 0) {
    y += 180;
  } else {
    y -= 180;
  }

  Serial.print("AngleX= ");
  Serial.println(x);
  Serial.print("AngleY= ");
  Serial.println(y);
  Serial.print("AngleZ= ");
  Serial.println(z);

 Serial.println("-----------------------------------");
     delay(400);
}
```

# Annexe 3



Code du capteur vitesse :

```
// Déclaration des broches
const int capteurPin = 2; // Broche utilisée pour le capteur magnétique
unsigned long tempsPrecedent = 0; // Pour stocker le temps précédent
volatile unsigned long pulses = 0; // Pour stocker le nombre de pulses détectés

// Fonction pour compter les pulses
void compterPulses() {
  pulses++; // Incrémenter le compteur de pulses
}

void setup() {
  Serial.begin(9600); // Initialisation de la communication série
  pinMode(capteurPin, INPUT); // Définir la broche du capteur comme une entrée
  attachInterrupt(digitalPinToInterrupt(capteurPin), compterPulses, RISING); // Attacher l'interruption au capteur
}

void loop() {
  unsigned long tempsActuel = millis(); // Obtenir le temps actuel en millisecondes

  // Calculer la vitesse en tr/min (nombre de pulses par minute)
  float vitesse = (float)pulses * 60000 / (float)(tempsActuel - tempsPrecedent);

  Serial.print("Vitesse (tr/min): ");
  Serial.println(vitesse);

  pulses = 0; // Réinitialiser le compteur de pulses
  tempsPrecedent = tempsActuel; // Mettre à jour le temps précédent


}
```

# Annexe 4



## Code Arduino pour que ESP 32 envoie les données du capteur d'effort :

```
// Include des bibliothèques nécessaires
#include <WiFi.h>

// Déclaration des informations de connexion WiFi
const char* ssid = "Nom_du_WiFi";
const char* password = "Mot_de_passe_du_WiFi";

// Paramètres de l'adresse IP et du port pour la communication série
const IPAddress address = IPAddress(192, 168, 1, 100); // Adresse IP de l'Arduino
const uint16_t port = 9600; // Port série de l'Arduino

// Déclaration des broches
const int capteurPin = A0; // Broche analogique pour le capteur d'effort

// Variables pour stocker les données du capteur
int valeurCapteur = 0;

// Déclaration d'un objet client pour la communication série
WiFiClient client;

void setup() {
  Serial.begin(9600); // Initialisation de la communication série
  delay(100);

  // Connexion au réseau WiFi
  Serial.println();
  Serial.print("Connexion au réseau WiFi ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connecté");
}

void loop() {
  // Lecture de la valeur du capteur d'effort
  valeurCapteur = analogRead(capteurPin);

  // Connexion au serveur Arduino
  if (!client.connected()) {
    if (client.connect(address, port)) {
      Serial.println("Connexion au serveur Arduino établie");
    }
  }

  // Envoi des données au serveur Arduino
  if (client.connected()) {
    client.print(valeurCapteur);
    client.print("\n"); // Ajouter un saut de ligne pour indiquer la fin des données
  }

  delay(1000); // Attendre 1 seconde avant d'envoyer la prochaine valeur
}
```

# Annexe 5



Code Arduino pour le potentiométre :

```arduino
const int potPin = A0; // Broche analogique connectée au potentiomètre

void setup() {
  Serial.begin(9600); // Initialisation de la communication série
}

void loop() {
  // Lecture de la valeur du potentiomètre
  int valeurPotentiometre = analogRead(potPin);

  // Normalisation de la valeur entre 0 et 1
  float valeurNormalisee = valeurPotentiometre / 1023.0;

  // Affichage de la valeur normalisée
  Serial.print("Valeur normalisée: ");
  Serial.println(valeurNormalisee);

  delay(100); // Attendre un court instant avant de lire à nouveau la valeur
}
```

# Annexe 6

54

## Code du moteur :

```
#include <HX711_ADC.h>
#include <SoftwareSerial.h>
#include <Stepper.h>
#include "Wire.h"

const int potPin = A3;  // Broche analogique où le potentiomètre est connecté
float potValue;         // Variable pour stocker la valeur lue du potentiomètre
float scaledValue;      // Variable pour stocker la valeur mise à l'échelle

// Constants for HX711
const int HX711_dout = 4; // ESP > HX711 dout pin
const int HX711_sck = 5;  // ESP > HX711 sck pin
HX711_ADC LoadCell(HX711_dout, HX711_sck);
const int calVal_eepromAdress = 0;
unsigned long t = 0;

// Constants for load cells
const float L = 20.0; // Length in cm
float F1 = 0;
float F2 = 0;
bool newData1 = false;
bool newData2 = false;

// Constants for MPU6050
const int MPU_addr = 0x68;
int16_t AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ;
int minVal = 265;
int maxVal = 402;
double x, y, z;

// Constants for ratios
float r4 = 41 / 17;
float r3 = 41 / 19;
float r2 = 41 / 21;
float r1 = 41 / 24;

// Constants for physical properties
float m = 70;
float ax = 20;
float g = 9.8;
float R = 0.33;
float f = 0.00739;
float k = 0.292;
float Vv = 0;
float n = 0.78;
float r;
float v = 0;
float Fr;
float Cp = 0;
int targetPosition;
float steps;

// Constants for stepper motor
const int stepsPerRevolution = 200;
Stepper myStepper(stepsPerRevolution, 8, 9, 10, 11);
const int positionValues[] = {1, 2, 3, 4};
const float valeurpas[] = {63.7, 63.7, 63.7};
int currentPosition = 0;

// Constants for wind sensors
const int windSensor1Pin = A0;
const int windSensor2Pin = A1;
// Calibration values
float vitesseEtalonage1 = 9.4;
float tensionEtalonage1 = 0.36;
float vitesseEtalonage2 = 9.4;
float tensionEtalonage2 = 1.23;

// Constants for IR sensor and LED
const int irPin = 8;
const int ledPin = 9;
int irValue = HIGH;
int previousIrState = HIGH;
int countFallingEdges = 0;
```

```
unsigned long samplingDuration = 2000;

// Constants for transistor control pin
const int transistorPin = 12;
void moveStepper(int targetPosition) {
  if (targetPosition < 0 || targetPosition >= stepsPerRevolution) {
    Serial.println("Position invalide.");
    return;
  }

  Serial.print("Déplacement vers la position: ");
  Serial.println(targetPosition);

  while (currentPosition != targetPosition) {
    if (currentPosition < targetPosition) {
      steps = valeurpas[currentPosition];
      myStepper.step(steps);
      currentPosition++;
    } else {
      steps = valeurpas[currentPosition - 1];
      myStepper.step(-steps);
      currentPosition--;
    }

    delay(10);
    Serial.print("Position actuelle: ");
    Serial.println(currentPosition);
    Serial.print("rapport: ");
    Serial.println(positionValues[currentPosition]);
  }

  // Set transistor pin HIGH when motor stops
  digitalWrite(transistorPin, HIGH);
}
void setup() {
  // Serial communication initialization
  Serial.begin(9600);
  // HX711 setup
  LoadCell.begin();
  float calibrationValue = -35.97;
  unsigned long stabilizingtime = 2000;
  boolean _tare = true;
  LoadCell.start(stabilizingtime, _tare);
  if (LoadCell.getTareTimeoutFlag()) {
    while (1);
  } else {
    LoadCell.setCalFactor(calibrationValue);
  }
  // MPU6050 setup
  Wire.begin();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);
  Wire.write(0);
  Wire.endTransmission(true);
  // Stepper motor setup
  myStepper.setSpeed(60);
  // IR sensor setup
  pinMode(ledPin, OUTPUT);
  pinMode(irPin, INPUT);
  pinMode(transistorPin, OUTPUT);
  digitalWrite(transistorPin, LOW);
}

void loop() {

  // HX711 data reading
  static boolean newDataReady = 0;
  if (LoadCell.update()) newDataReady = true;

  if (newDataReady) {
    float M = LoadCell.getData();
    float F = M / 100;
    Cp = F*L;
    newDataReady = 0;
  }
```

```
    float M = LoadCell.getData();
    float F = M / 100;
    Cp = F*L;
    newDataReady = 0;
  }

  // Read data from ESP1
  if (Serial1.available() > 0) {
    M1 = Serial1.parseFloat();
    F1=M1/100;
    newData1 = true;
  }

  // Read data from ESP2
  if (Serial2.available() > 0) {
    M2 = Serial2.parseFloat();
    F2=M2/100;
    newData2 = true;
  }

  // Calculate Cp when data from both ESPs is available
  if (newData1 && newData2) {
    float Cp = (F1 - F2) * L;

    // Reset flags
    newData1 = false;
    newData2 = false;
  }

void loop() {
  // MPU6050 data reading
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x3B);
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr, 14, true);
  AcX = Wire.read() << 8 | Wire.read();
  AcY = Wire.read() << 8 | Wire.read();
  AcZ = Wire.read() << 8 | Wire.read();
```

```

  int xAng = map(AcX, minVal, maxVal, -90, 90);
  int yAng = map(AcY, minVal, maxVal, -90, 90);
  int zAng = map(AcZ, minVal, maxVal, -90, 90);

  x = RAD_TO_DEG * (atan2(-yAng, -zAng) + PI);
  y = RAD_TO_DEG * (atan2(-xAng, -zAng) + PI);
  z = RAD_TO_DEG * (atan2(-yAng, -xAng) + PI);

  if (x < 0) {
    x += 180;
  } else {
    x -= 180;
  }

  if (y < 0) {
    y += 180;
  } else {
    y -= 180;
  }

  // Wind sensor data reading
  float tensionArduino1 = tensionEtalonage1 * (1023.0 / 5.0);
  float vitesse1 = analogRead(windSensor1Pin) * (vitesseEtalonage1 / tensionArduino1);

  float tensionArduino2 = tensionEtalonage2 * (1023.0 / 5.0);
float vitesse2 = analogRead(windSensor2Pin) * (vitesseEtalonage2 / tensionArduino2);

  float vitesse1KmH = vitesse1 * 3.6;
  float vitesse2KmH = vitesse2 * 3.6;

  Vv = vitesse1KmH - vitesse2KmH;

  // IR sensor and speed calculation
  unsigned long startTime = millis();

  while (millis() - startTime < samplingDuration) {
    irValue = digitalRead(irPin);
```

```
221
222   if (irValue == LOW && previousIrState == HIGH) {
223     countFallingEdges++;
224   }
225
226   previousIrState = irValue;
227
228   if (irValue == HIGH) {
229     digitalWrite(ledPin, HIGH);
230   } else {
231     digitalWrite(ledPin, LOW);
232   }
233 }
234
235 double vitesseAngulaire = (countFallingEdges) * 60000.0 / (double)(samplingDuration) * 8;
236 double vitesseLineaire = vitesseAngulaire * (2 * PI * 1) / 60;
237 double vitesseKmH = vitesseLineaire * 3.6;
238
239 v = vitesseKmH;
240 if (v != 0) {
241   potValue = analogRead(potPin); // Lire la valeur du potentiomètre (0-1023)
242   scaledValue = potValue * (1.389 / 1023.0); // Mettre la valeur à l'échelle de 0 à 1.389
243
244   ax = scaledValue
245 } else {
246   ax= 1
247 }
248 // Friction force calculations
249 if (Vv >= 0) {
250   Fr = m * g * sin(x * (PI / 180)) + f * m * g * cos(x * (PI / 180)) + k * (Vv * Vv);
251 } else {
252   Fr = m * g * sin(x * (PI / 180)) + f * m * g * cos(x * (PI / 180)) - k * (Vv * Vv);
253 }
254
255 r = (Cp * n) / ((m * ax + Fr) * R);
256
257 // Determine target position
258 if (Cp != 0 && v != 0 && r <= r1) {
```

```
259     targetPosition = 0;
260   }
261   if (Cp != 0 && v != 0 && r <= r2 && r > r1) {
262     targetPosition = 1;
263   }
264   if (Cp != 0 && v != 0 && r <= r3 && r > r2) {
265     targetPosition = 2;
266   }
267   if (Cp != 0 && v != 0 && r > r3) {
268     targetPosition = 3;
269   }
270   if (Cp != 0 && v == 0) {
271     targetPosition = 0;
272   }
273   if (Cp == 0 && v > 21.16) {
274     targetPosition = 3;
275   }
276   if (Cp == 0 && 21.16 >= v && v > 17.44) {
277     targetPosition = 2;
278   }
279   if (Cp == 0 && 17.44 >= v && v > 13.79) {
280     targetPosition = 1;
281   }
282   if (Cp == 0 && 13.79 >= v) {
283     targetPosition = 0;
284   }
285
286   // Move stepper motor
287   moveStepper(targetPosition);
288
289   // Display current position and step value
290   Serial.print("Position actuelle: ");
291   Serial.println(currentPosition);
292   Serial.print("rapport: ");
293   Serial.println(positionValues[currentPosition]);
294   Serial.print("-----------------------------\n");
295
296   delay(1);
297 }
```

# Annexe 7



Code Arduino pour l'écran LCD :

```arduino
#include <LiquidCrystal.h>

// Initialisation de l'écran LCD avec les broches correspondantes
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

// Déclaration des broches
const int capteurPin = 2; // Broche utilisée pour le capteur de vitesse
unsigned long tempsPrecedent = 0; // Pour stocker le temps précédent
volatile unsigned long pulses = 0; // Pour stocker le nombre de pulses détectés

// Fonction pour compter les pulses
void compterPulses() {
  pulses++; // Incrémenter le compteur de pulses
}

void setup() {
  // Initialisation de la communication série
  Serial.begin(9600);

  // Initialisation de l'écran LCD
  lcd.begin(16, 2);
  lcd.print("Vitesse: ");

  // Configuration de la broche du capteur
  pinMode(capteurPin, INPUT);
  attachInterrupt(digitalPinToInterrupt(capteurPin), compterPulses, RISING); // Attacher l'interruption au capteur
}

void loop() {
  unsigned long tempsActuel = millis(); // Obtenir le temps actuel en millisecondes

  // Calculer la vitesse en tr/min (nombre de pulses par minute)
  float vitesse = (float)pulses * 60000 / (float)(tempsActuel - tempsPrecedent);

  // Afficher la vitesse sur l'écran LCD
  lcd.setCursor(0, 1); // Positionner le curseur sur la deuxième ligne
  lcd.print("        "); // Effacer l'ancienne valeur
  lcd.setCursor(0, 1); // Positionner le curseur à nouveau
  lcd.print(vitesse);

  // Réinitialiser les variables
  pulses = 0; // Réinitialiser le compteur de pulses
  tempsPrecedent = tempsActuel; // Mettre à jour le temps précédent

}
```