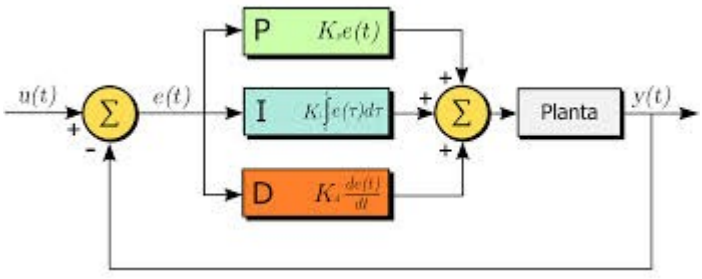
Asservissement moteur CC

Table des matières

Lorsqu'un moteur entraîne un robot pour qu'il se déplace d'une certaine distance, l'organe de commande va envoyer une consigne pendant une certaine durée, sans savoir si le moteur aura réellement effectué ce qu'on lui a demandé.
L'asservissement consiste à vérifier ce que le moteur a réellement effectué et à corriger si nécessaire (en augmentant sa commande, pour rattraper le retard par exemple).



1. Introduction

Le moteur DC fonctionne quand on lui envoie un courant. Le rotor tourne plus ou moins vite selon celui-ci, puis s'arrête dès lors qu'il n'est plus alimenté. Cependant, on ne connaît ni la distance angulaire parcourue, ni la vitesse de rotation. Aussi, si l'on veut réaliser un asservissement en vitesse ou en position, il faut transformer la consigne pour envoyer un courant pendant le temps nécessaire pour atteindre l'objectif. Il faut pouvoir mesurer ce qu'il se passe pendant que le moteur est en marche, calculer une position ou une vitesse, en déduire une erreur, puis la corriger pour atteindre la consigne de sortie. Il faut donc se servir de codeurs incrémentaux, acquérir les données, faire les calculs et commander le moteur en même temps.

2. Les interruptions logicielles

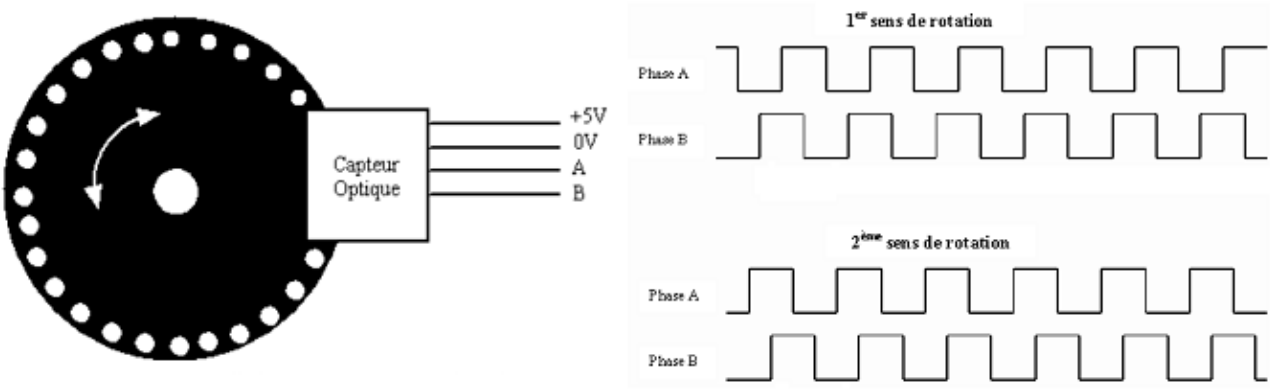
Pour compter le nombre de tour pendant que le moteur tourne, on utilise une interruption qui va se déclencher à chaque transition de l'encodeur et viendra simplement incrémenter un compteur. L'encodeur est intégré et constitué de deux capteurs hall (équivalents magnétique des capteurs optique) qui permettent de lire ces interruptions sous forme de front montants et descendants. Deux capteurs, deux fonctions d'interruptions.

Pour obtenir de l'aide, cliquer sur le bouton 

3. Acquisition des impulsions de la roue codeuse

Les roues codeuses sont essentielles à l'asservissement du moteur. Elles permettent à terme de connaître le sens ainsi que la vitesse de rotation du moteur. Un disque magnétique passe devant deux capteurs qui le détectent tour à tour. A chaque passage de l'aimant, une impulsion est envoyée à la carte Arduino. Il y a un front montant et un front descendant pour chaque capteur, soit quatre positions différentes possible (HH,BB,HB,BH). Sur une période, il est donc possible de passer par quatre positions équidistantes de 90° chacune, 360° équivalent à une période. C'est pour cela que l'on parle de décalage en quadrature du signal. De plus, selon l'enchaînement des fronts, il est facile de déterminer le sens de rotation. Par exemple si on avait enregistré HH et qu'ensuite HB provient, on sait que le moteur tourne dans le sens horaire. Si au contraire BH provient, c'est l'inverse.

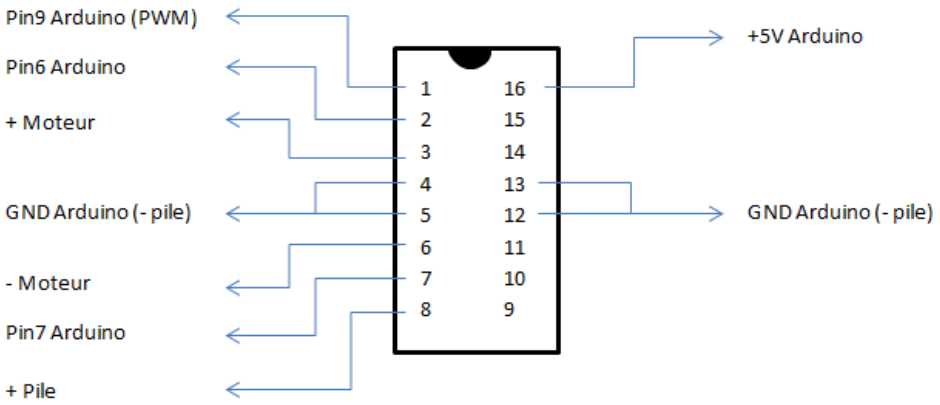
Le codeur magnétique est moins précis qu'un codeur optique. En effet, il utilise des aimants à la place des traditionnels traits noirs, et ceux-ci doivent éviter d'être trop proches pour ne pas créer d'interférences. Cependant, fixé à l'arbre moteur, le nombre d'impulsions par tour de l'arbre de sortie est multiplié par le réducteur et peut donc être largement suffisant. L'image ci-dessous montre le fonctionnement d'un codeur optique, mais il suffit de remplacer virtuellement les points blancs par des aimants pour obtenir un codeur magnétique.



Tirer des informations du codeur incrémental revient à créer une fonction compteur. On enregistre une valeur que l'on vient incrémenter ou décrémenter selon l'ordre des impulsions de la quadrature. Remarque : il est possible de connaître la vitesse de rotation à l'aide d'un seul capteur. Mais le fait d'utiliser deux capteurs permet de plus d'en déterminer le sens.

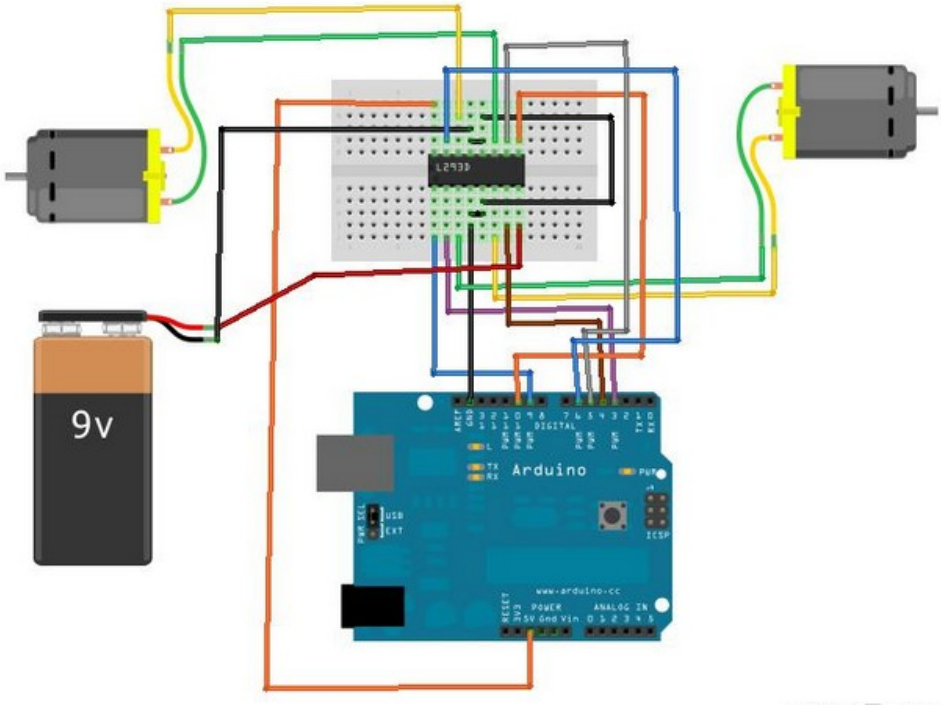
4. Commande et alimentation du moteur

Pour commander le moteur dans les deux sens, on utilise un pont en H intégré de type L293D (système composé de quatre transistors qui permet de contrôler électroniquement le sens du courant ou de la tension).



L293D

Pour obtenir de l'aide, cliquer sur le bouton 



montage

5. Échantillonnage

L'échantillonnage est une technique utilisée pour le traitement du signal et permet de dire à l'Arduino de faire des calculs tous les intervalles donnés afin de ne pas saturer la puce de calculs inutiles. Ainsi, au lieu de faire des mesures continues et de ralentir le microprocesseur, celui-ci effectuera ses calculs par exemple toutes les 50ms.

Ainsi, toutes les 50ms, l'Arduino refera les calculs et nous dira si oui ou non on s'est éloigné de la consigne d'entrée. Cela permettra alors de corriger la sortie.

```
#include <SimpleTimer.h>

SimpleTimer timer;
unsigned int time = 0;
const int frequence_echantillonnage = 20;

void setup()
{
      // toutes les 100ms, on appelle la fonction "asservissement"
      timer.setInterval(1000/frequence_echantillonnage, asservissement);
}
```

La carteArduino met en mémoire les impulsions de la roue codeuse en continu dans les interruptions, mais ne procédera à l'asservissement que toutes les 50ms.

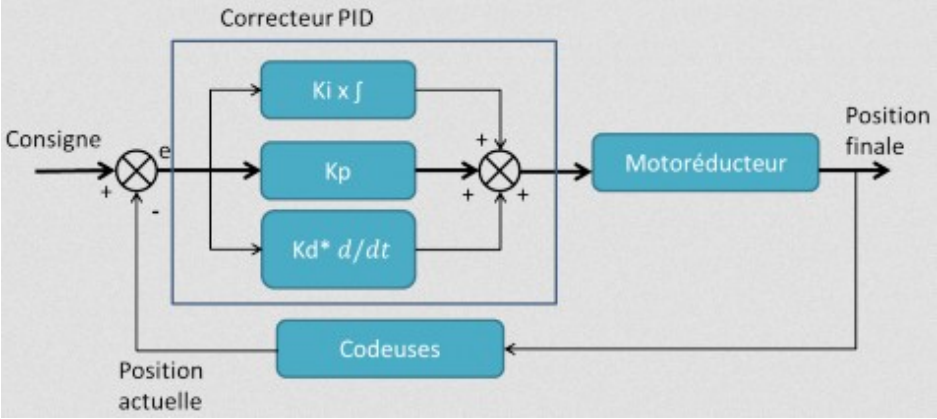
6. Asservissement

L'asservissement est réalisé à l'aide un correcteur PID (à effets proportionnel, intégral, et dérivé). Cet asservissement peut être modélisé par la formule ci-dessous, où V est la consigne (en Tr/s) à envoyer au moteur, et Ki, Kd, Kp des coefficients à modifier à la main jusqu'à trouver les coefficients les plus satisfaisants :

$$V = Kp \times Erreur + Kd \times \Delta Erreur + KI \times \int Erreur$$

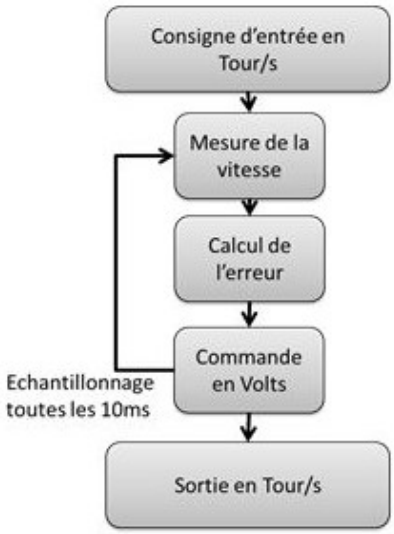
avec :

- Erreur = Consigne – Nombre Tour roue par seconde mesuré
- Δ Erreur = Erreur – Erreur précédente
- $\int$ Erreur = $\sum$ erreurs



Pour obtenir de l'aide, cliquer sur le bouton 

6.1. Asservissement en vitesse



Voici la fonction « asservissement », appelée par la fonction d'échantillonnage :

```
void asservissement()
{
    int frequence_codeuse = frequence_echantillonnage *
tick_codeuse; //100*tick_codeuse (pour obtenir des
secondes)
    float vit_roue_tour_sec = (float)frequence_codeuse /
(float)tick_par_tour_codeuse / (float)rapport_reducteur; //
(100*tick_codeuse)/32/19
    float erreur = consigne_moteur - vit_roue_tour_sec; //
pour le proportionnel
    somme_erreur += erreur; // pour l'intégrateur
    float delta_erreur = erreur - erreur_precedente; //
pour le dérivateur
    erreur_precedente = erreur; // P : calcul de la
commande
    vitMoteur = kp*erreur + ki*somme_erreur +
kd*delta_erreur; //somme des trois erreurs
}
```

En donnée brute, on obtient un nombre d'impulsions par période d'échantillonnage. Il n'y a en fait qu'à convertir cette donnée dans l'unité de la consigne et d'en faire la différence pour trouver l'erreur.

Ici, la consigne est en tours/s. On n'utilise qu'un des deux capteurs hall, soit 32 impulsions par tour et la réduction est d'1/19eme. Associée à une roue, on peut obtenir des m/s ou des km/h à partir du diamètre.

Le code source :

```
// asservissement d'un moteur CC en vitesse avec un correcteur PID.

#include <SimpleTimer.h>

SimpleTimer timer;                  // Timer pour échantillonnage

unsigned int tick_codeuse = 0;      // Compteur de tick de la codeuse
int vitMoteur = 0;                          // Commande du moteur
const int frequence_echantillonnage = 100; // Fréquence d'exécution de l'asservissement
const int rapport_reducteur = 19;           // Rapport nombre de tours de l'arbre moteur
et de la roue
const int tick_par_tour_codeuse = 32;  //64 tick sur deux capteurs hall, ici un seul
capteur

//definition des entrées
const int pinInput1 = 6;          // Commande de sens moteur, Input 1
const int pinInput2 = 7;          // Commande de sens moteur, Input 2
const int pinPower  = 9;          // Commande de vitesse moteur, Output Enabled1

//consigne en tour/s
const float consigne_moteur = 2;  // Consigne nombre de tours de roue par seconde

// init calculs asservissement PID
float erreur_precedente = consigne_moteur; // (en tour/s)
float somme_erreur = 0;

//Definition des constantes du correcteur PID
const float kp = 200;       // Coefficient proportionnel (choisis par essais successifs)
const float ki = 5;         // Coefficient intégrateur
const float kd = 100;       // Coefficient dérivateur

/* Routine d'initialisation */
void setup()
{
    Serial.begin(115200);          // Initialisation port COM
    pinMode(pinPower, OUTPUT);     // Sorties commande moteur
    pinMode( pinInput1, OUTPUT );
    pinMode( pinInput2, OUTPUT );

    analogWrite(pinPower, 0);  // Initialisation sortie moteur à 0
    delay(300);                // Pause de 0,3 sec pour laisser le temps au moteur de
s'arréter si celui-ci est en marche

    // Interruption sur tick de la codeuse  (interruption 0 = pin2 arduino)
    attachInterrupt(0, compteur, CHANGE);

    // Interruption pour calcul du PID et asservissement appelee toutes les 10ms
    timer.setInterval(1000 / frequence_echantillonnage, asservissement);
}

/* Fonction principale */
void loop()
{
    timer.run();  //on fait tourner l'horloge
    delay(10);
}

//---- Interruption sur tick de la codeuse
void compteur()
{
    tick_codeuse++;
    // On incrémente le nombre de tick de la codeuse.   un seul sens
}

//---- Interruption pour calcul du P
```

```
void asservissement()
{
    // Calcul de l'erreur
    int frequence_codeuse = frequence_echantillonnage * tick_codeuse; //100*tick_codeuse
    float vit_roue_tour_sec = (float)frequence_codeuse / (float)tick_par_tour_codeuse /
(float)rapport_reducteur;    //(100*tick_codeuse)/32/19
    float erreur = consigne_moteur - vit_roue_tour_sec; // pour le proportionnel
    somme_erreur += erreur; // pour l'intégrateur
    float delta_erreur = erreur - erreur_precedente;  // pour le dérivateur
    erreur_precedente = erreur;

    // Réinitialisation du nombre de tick de la codeuse
    tick_codeuse = 0;

    // P : calcul de la commande
    vitMoteur = kp*erreur + ki*somme_erreur + kd*delta_erreur;  //somme des trois erreurs

    // Normalisation et contrôle du moteur
    if ( vitMoteur > 255 )
    {
        vitMoteur = 255;  // sachant que l'on est branché sur un pont en H L293D
    }
    else if ( vitMoteur < 0 )
    {
        vitMoteur = 0;
    }

    TournerDroite (vitMoteur);

    // DEBUG
    Serial.print(vit_roue_tour_sec, 8);  // affiche à gauche la vitesse et à droite
l'erreur
    Serial.print(" : ");
    Serial.print(erreur, 4);
    Serial.print(" : ");
    Serial.print(vitMoteur);
    Serial.println();
}

//---- fonction pour faire tourner le moteur dans un sens (a droite)
void TournerDroite( int powerRate )
{
    digitalWrite(pinInput1, LOW);
    digitalWrite(pinInput2, HIGH);
    analogWrite(pinPower, powerRate);
}
```

6.2. Asservissement en position

L'asservissement en position est un peu plus compliqué à mettre en place. Une chose importante est d'effectuer tous les calculs en nombres entiers pour ne pas perdre d'informations.

De plus, plus on se rapproche de l'objectif, moins la vitesse du moteur est élevée. On peut bien sûr jouer avec les constantes du PID, pour trouver l'optimum entre vitesse, dépassement et temps de réaction.

Pour les nombres relatifs présents, ils sont encore une fois le fruit de conversions logiques. Par exemple, une commande de 10 degré en sortie équivaut à :

- 190 degrés en entrée (R=19)
- 0,52 tours d'arbre moteur (/365°)

- 33,28 ticks (64 ticks par tour)

Remarques : on a pris 5,23 cm pour un tour de la roue fixée au moteur et par conséquent il faut 68,8° pour faire avancer la roue d'un cm (rapport calculé entre l'angle de consigne et la distance parcourue mesurée à la règle).

Le code source :

```
// asservissement en position angulaire un moteur à courant continu.

#include <SimpleTimer.h>

SimpleTimer timer;                  // Timer pour échantillonnage

//definition des entrées
const int pinInput1 = 6;          // Commande de sens moteur, Input 1
const int pinInput2 = 7;          // Commande de sens moteur, Input 2
const int pinPower  = 9;          // Commande de vitesse moteur, Output Enabled1
const int encoderPinA = 2;        // compteur 1
const int encoderPinB = 3;        // compteur 2

//init echantillonage
unsigned int time = 0;
const int frequence_echantillonnage = 20;

//init compteur :
int encoder0Pos = 0; //position de départ=0
int lastReportedPos = 0;
boolean A_set = false;
boolean B_set = false;

//consigne
const double target_cm = 48;
double target_deg = 68.8 * target_cm;
int target_ticks; //plus simple d'asservir en ticks car ce sera toujours un nombre entier

// init calculs asservissement PID
int erreur = 0; //erreur
float erreurPrecedente = 0;
float somme_erreur = 0;

//Definition des constantes du correcteur PID
const float kp = 0.90;      // Coefficient proportionnel (choisis par essais successifs)
const float ki = 0;         // Coefficient intégrateur
const float kd = 0;         // Coefficient dérivateur

/* Routine d'initialisation */
void setup()
{

    target_ticks = target_deg / 360.0 * 19.0 * 64.0;

    Serial.begin(115200);          // Initialisation port COM

    pinMode(pinPower, OUTPUT);    // Sorties commande  moteur
    pinMode(pinInput1, OUTPUT);
    pinMode(pinInput2, OUTPUT);

    pinMode(encoderPinA, INPUT);  //sorties encodeur
    pinMode(encoderPinB, INPUT);
    digitalWrite(encoderPinA, HIGH);  // Resistance interne arduino ON
    digitalWrite(encoderPinB, HIGH);  // Resistance interne arduino ON

    // Interruption de l'encodeur A en sortie 0 (pin 2)
```

```
    attachInterrupt(0, doEncoderA, CHANGE);
    // Interruption de l'encodeur A en sortie 1 (pin 3)
    attachInterrupt(1, doEncoderB, CHANGE);

    analogWrite(pinPower, 0);  // Initialisation sortie moteur à 0
    delay(300);                // Pause de 0,3 sec pour laisser le temps au moteur de
s'arréter si celui-ci est en marche

    // Interruption pour calcul du PID et asservissement appelée toutes les 10ms
    timer.setInterval(1000 / frequence_echantillonnage, asservissement);
}

/* Fonction principale */
void loop()
{
    timer.run();  //on fait tourner l'horloge
}

//---- Cette fonction est appelée toutes les 20ms pour calcul du correcteur PID
void asservissement()
{
    time += 20; // pratique pour graphes excel après affichage sur le moniteur

    erreur = target_ticks - encoder0Pos;
    somme_erreur += erreur;

    // Calcul de la vitesse courante du moteur
    int vitMoteur = kp * erreur + kd * (erreur - erreurPrecedente) + ki * (somme_erreur);

    erreurPrecedente = erreur; // Ecrase l'erreur précedente par la nouvelle erreur

    // Normalisation et contrôle du moteur
    if ( vitMoteur > 255 ) vitMoteur = 255;  // on est branché sur un pont en H L293D
    else if ( vitMoteur < -255 ) vitMoteur = -255;

    Tourner (vitMoteur);

    float angle_deg = encoder0Pos / 19.0 / 64.0 * 360.0; //Position angulaire de sortie,
pratique pour comparer avec la consigne d'entrée
    float distance = encoder0Pos / 19.0 / 64.0 * 360.0 / 68.83;

    Serial.print(erreur);  // affiche sur le moniteur les données voulues
    Serial.print(" ");
    Serial.print(encoder0Pos);
    Serial.print(" ");
    Serial.print(angle_deg);
    Serial.print(" ");
    Serial.print(distance);
    Serial.print(" ");
    Serial.println(vitMoteur);
}

//---- Interruption appelée à tous les changements d'état de A
void doEncoderA()
{
    A_set = digitalRead(encoderPinA) == HIGH;

    encoder0Pos += (A_set != B_set) ? -1 : 1 ; //modifie le compteur selon les deux états
des encodeurs
}

//---- Interruption appelée à tous les changements d'état de B
void doEncoderB()
{
    B_set = digitalRead(encoderPinB) == HIGH;
```

```
    encoder0Pos += (A_set == B_set) ? -1 : 1 ; //modifie le compteur selon les deux états
des encodeurs
}

//---- Fonction appelée pour contrôler le moteur
void Tourner( int rapportCyclique )
{
    if ( rapportCyclique > 0 )
    {
        digitalWrite(pinInput1, LOW);
        digitalWrite(pinInput2, HIGH);
    }
    else
    {
        digitalWrite(pinInput1, HIGH);
        digitalWrite(pinInput2, LOW);
        rapportCyclique = -rapportCyclique;
    }

    analogWrite(pinPower, rapportCyclique);
}
```

7. Matériel utilisé

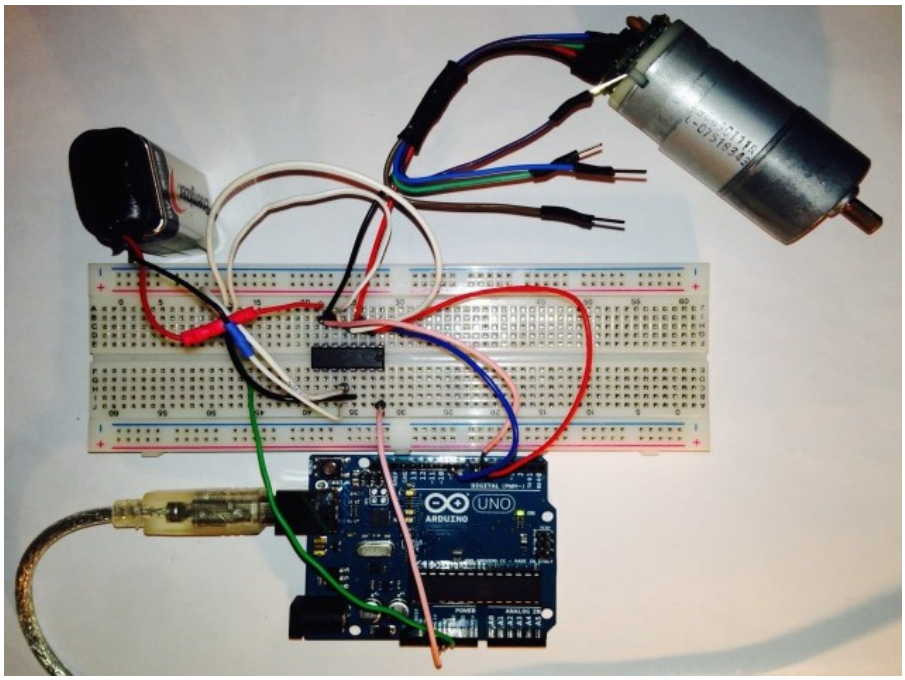
Liste du matériel :

- carte Arduino
- pont en H L293D
- moteur à courant continu 12V (de rapport de réduction 1/19) équipe d'une roue codeuse
- pile de 9V pour alimenter le moteur

Correspondances entre couleurs et rôles des différents câbles du moteur CC :

Color	Function
Black	motor power
Red	motor power
Blue	Hall sensor Vcc (3.5 – 20 V)
Green	Hall sensor GND
Yellow	Hall sensor A output
White	Hall sensor B output

Schéma du montage final :


UNO
ARDUINO