Remerciements

Remerciements

Vous tous
Vous toutes

Remerciements

Remerciements

Mathieu Lancry

Patrick Bonnin

Pierre Blazevic

Bases de l’électronique
Les actionneurs

La MCC
Machine à Courant Continu

La MCC

2 types :

- Machine à Courant Continu à aimant permanent
- Machine à Courant Continu à excitation indépendante

La MCC

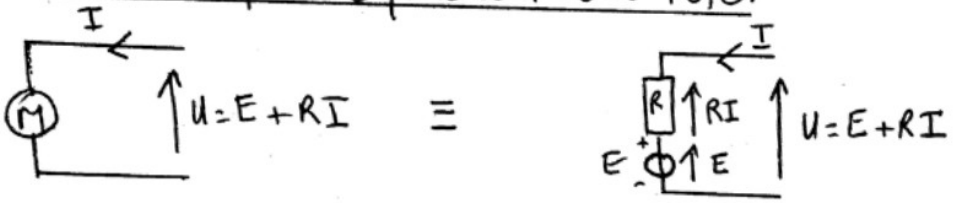
2 types :

- **Machine à Courant Continu à aimant permanent**
- Machine à Courant Continu à excitation indépendante

2 types :

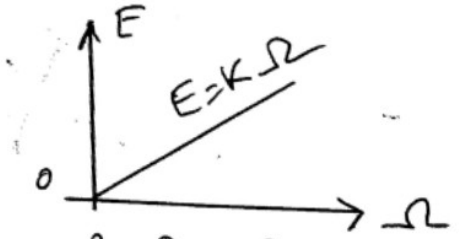
- Machine à Courant Conti
- Machine à Courant Conti

Modèle électrique équivalent du rotor



En moteur

Relation entre E et Ω



$$E = K\Omega$$

V V.rad⁻¹.s rad.s⁻¹

E : f.e.m en volts

Ω : vitesse angulaire en radians par seconde

Relation entre T_u et Ω

$P_{em} = EI = T_{em}\Omega$

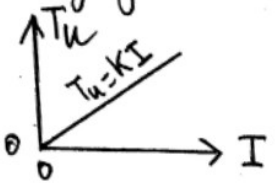
$$T_{em} = \frac{EI}{\Omega} = \frac{K\Omega I}{\Omega} = KI$$

or $T_u = T_{em} - T_p$ avec T_p : couple de pertes

Si le couple de pertes est négligeable devant le couple utile on a

$$T_u \simeq KI$$

N.m A



La MCC à aimants permanents

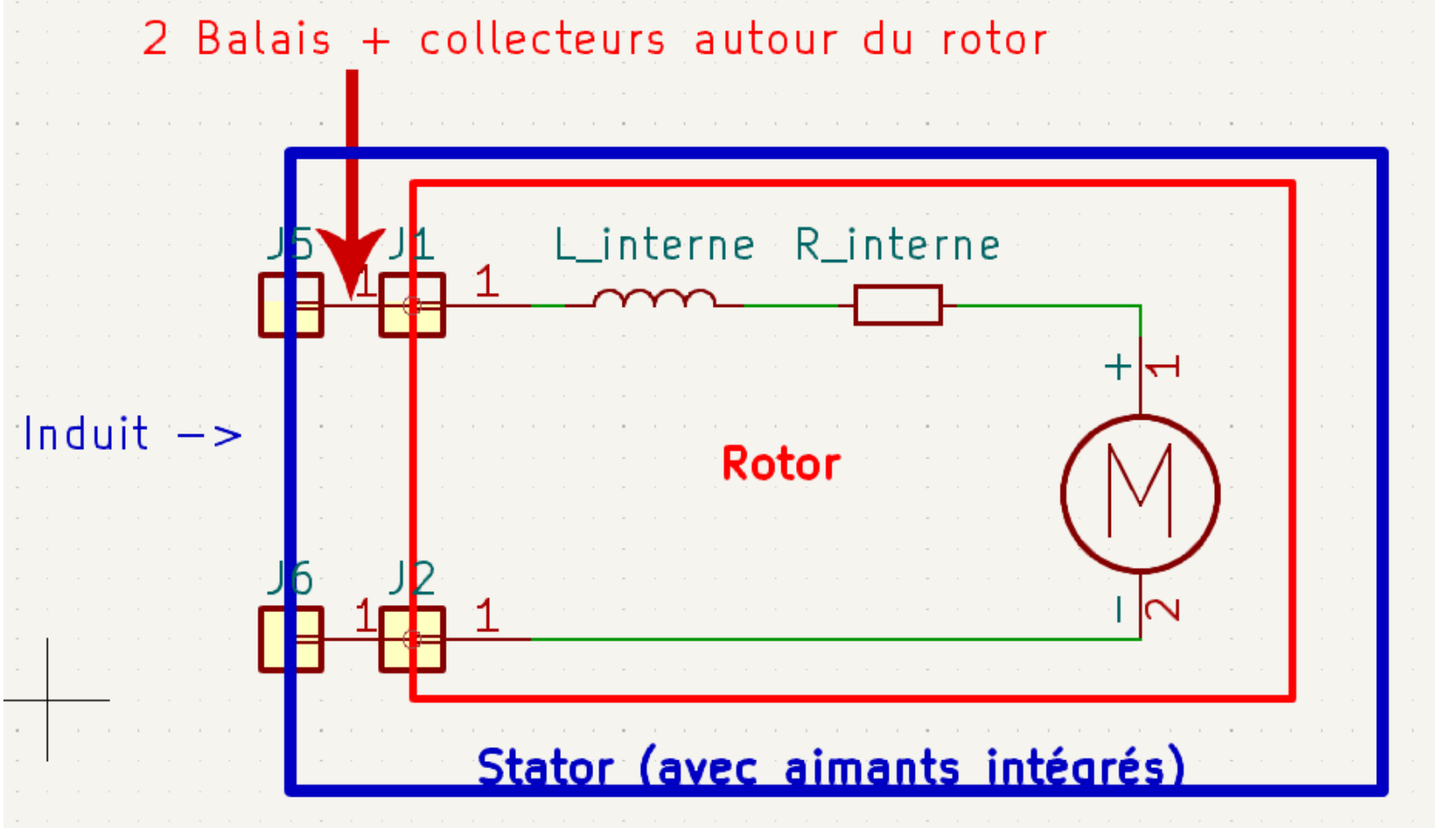
Attention aux connecteurs
→ Souder les 2 fils
→ Coller les fils sur la périphérie de la MCC

La MCC à aimants permanents

ɹrs

ɔériphérie de la MCC

La MCC à aimants permanents



La MCC

2 types :

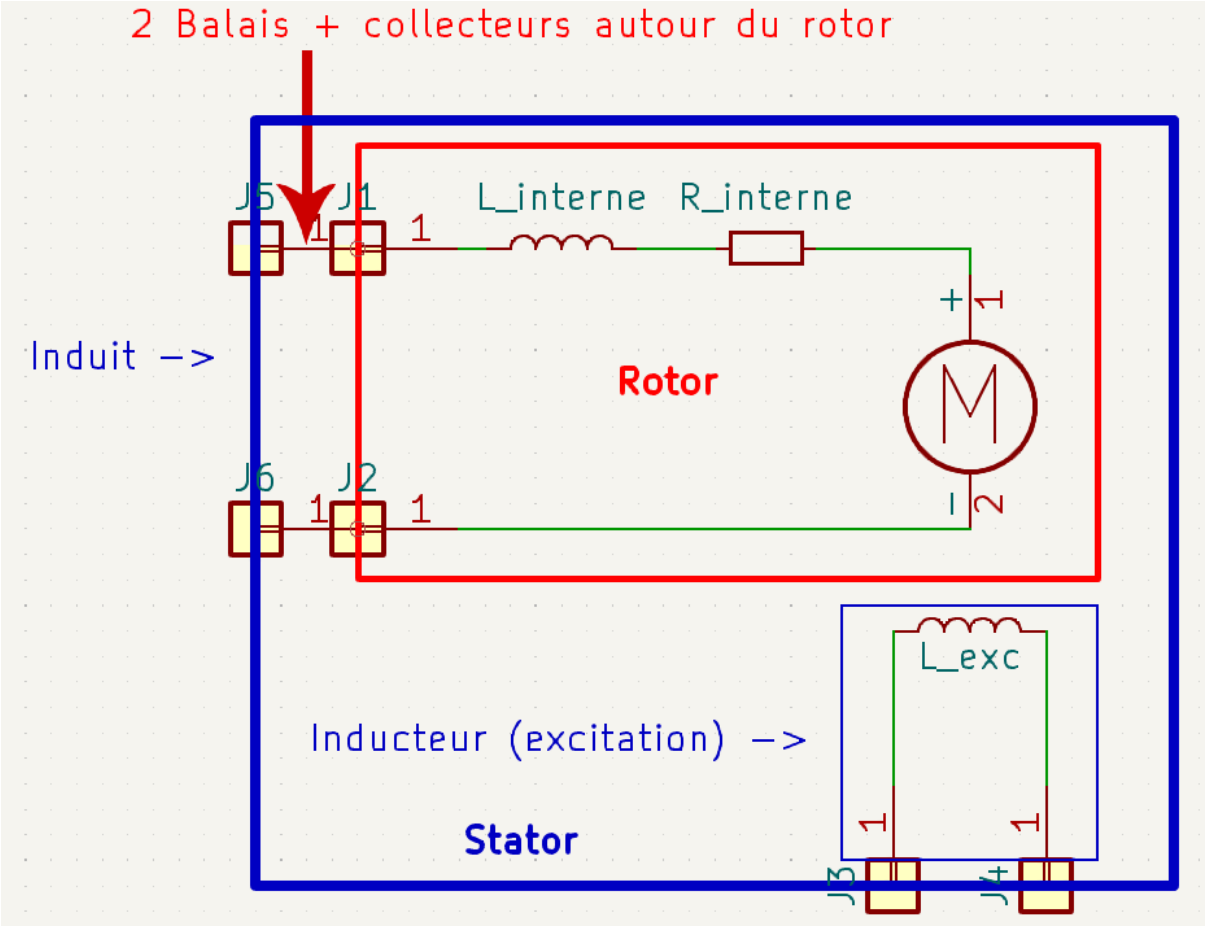
- Machine à Courant Continu à aimant permanent
- **Machine à Courant Continu à excitation indépendante**

La MCC à excitation indépendante

Induit

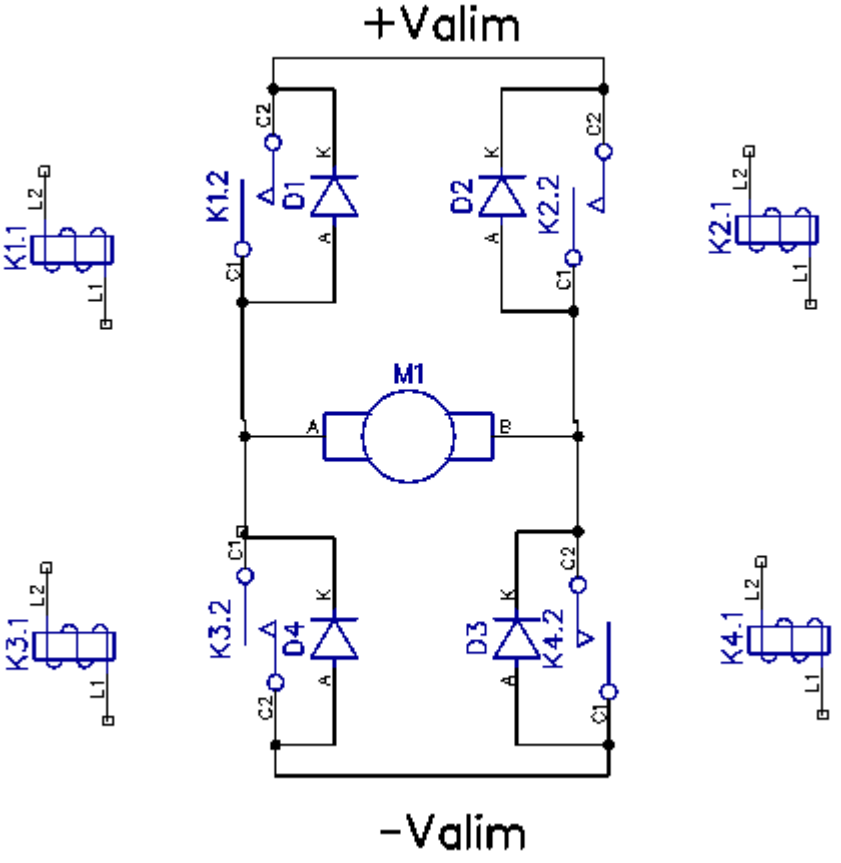
Inducteur
Excitation

La MCC à excitation indépendante



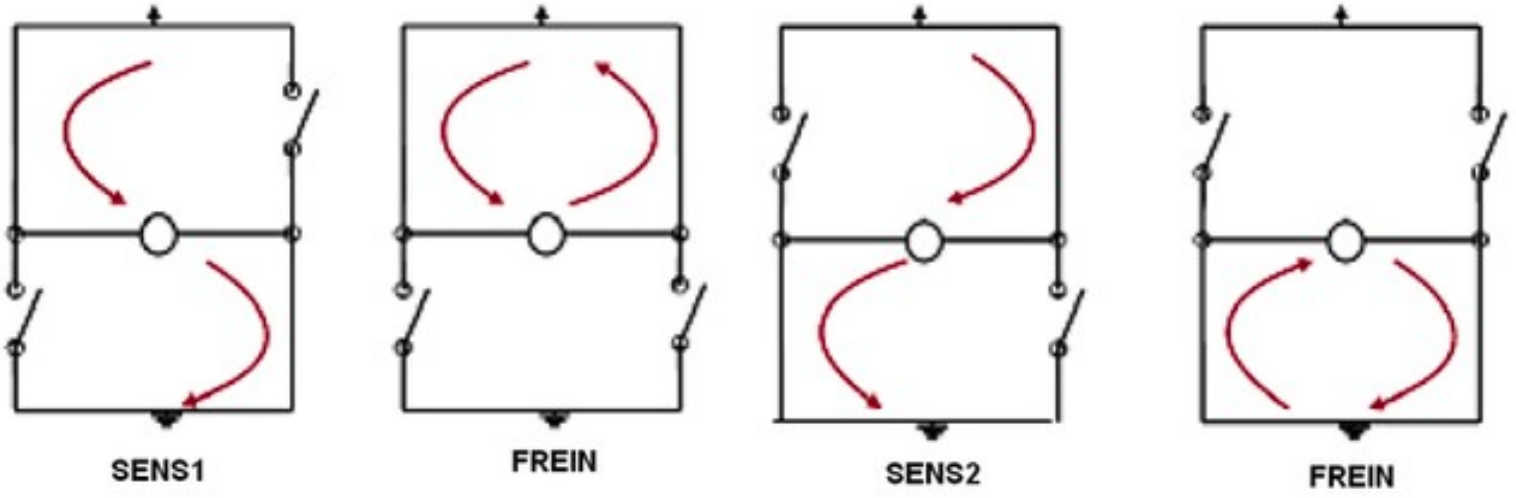
La MCC commandée par un µC

La MCC commandée par un µC



Le pont en H

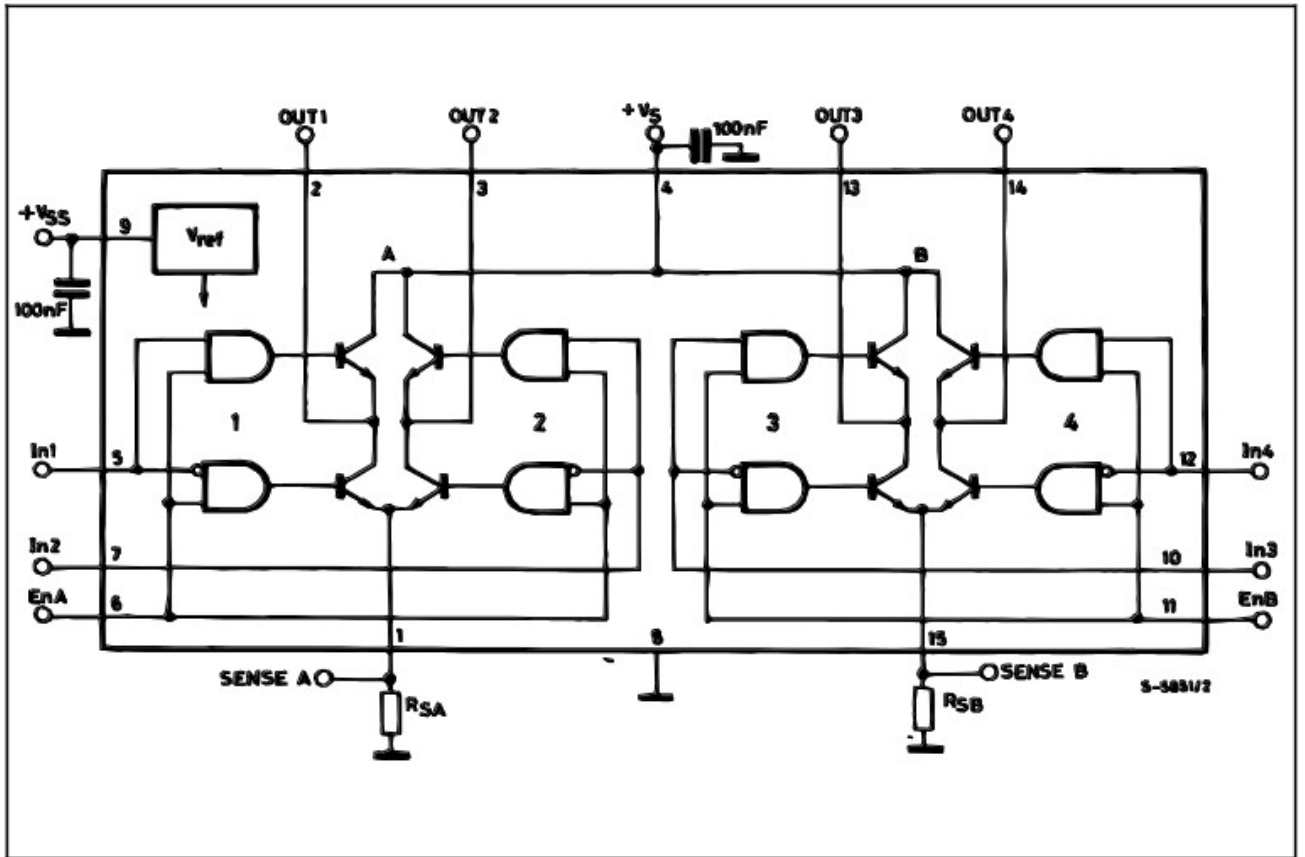
La MCC commandée par un µC



Le pont en H

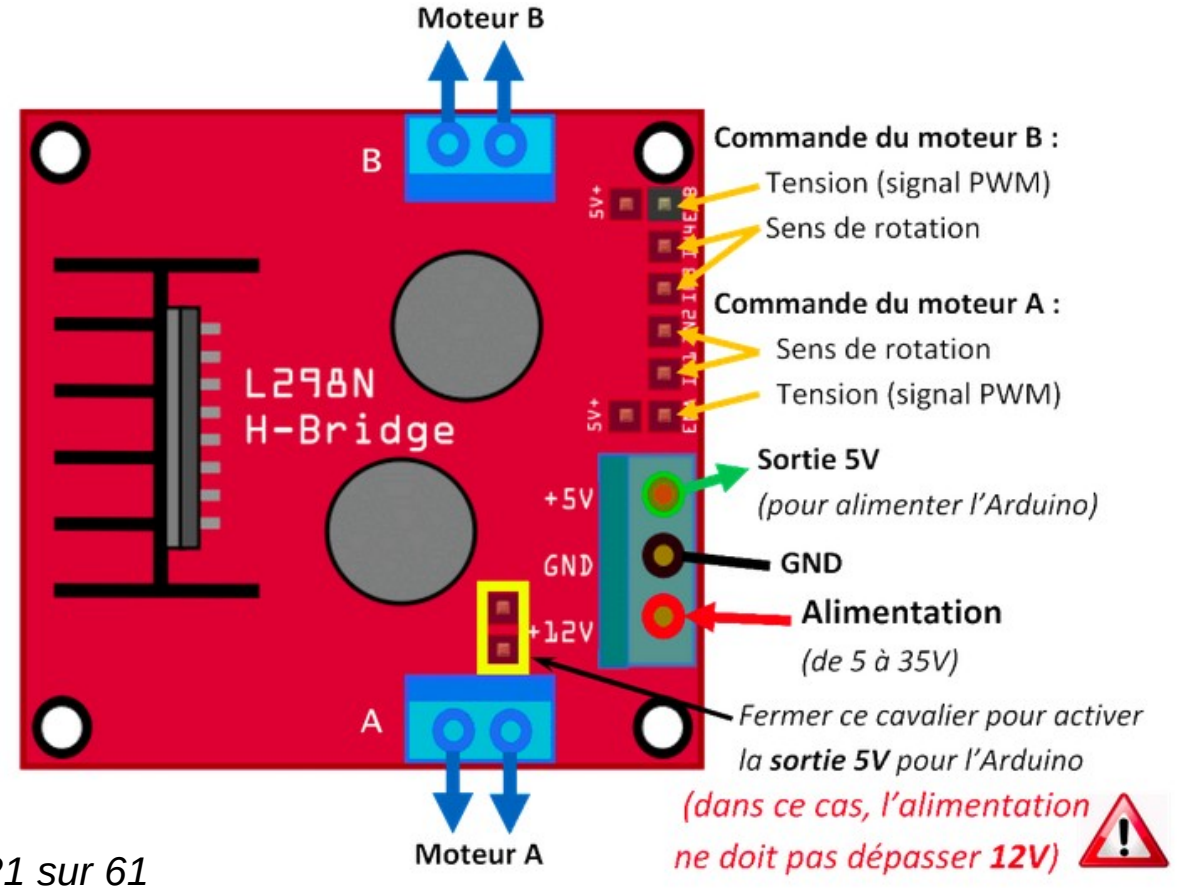
La MCC commandée par un µC

BLOCK DIAGRAM



L298

La MCC commandée par un µC

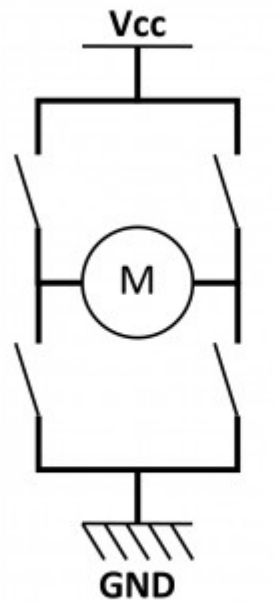
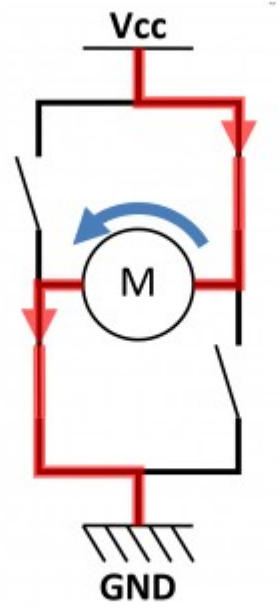
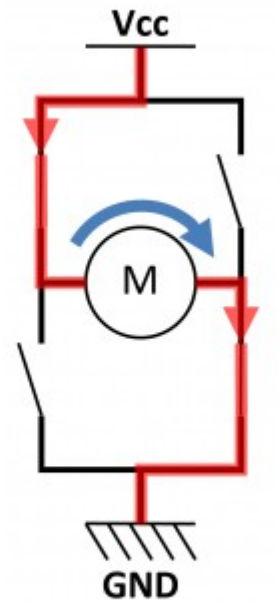
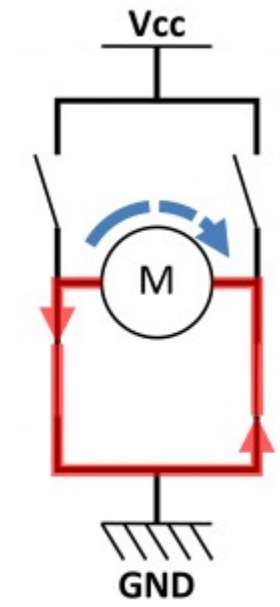


Le L298

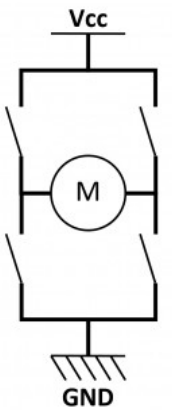
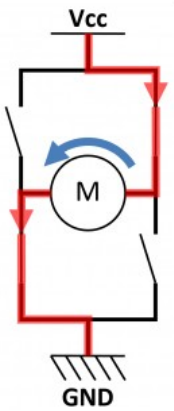
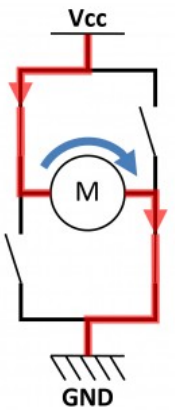
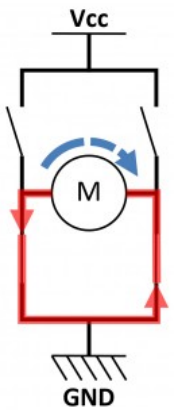
Datasheet du L298N

Courant max : 2A
Alimentation : 0V / 5V à 35V
2 MCC commandables par 1 µC
Sortie 5V pour le µC (si seul)

La MCC commandée par un µC

		Arrêt (moteur libre)	Sens +	Sens –	Arrêt (moteur freiné)
Moteur A	Moteur B				
In1	In3	LOW	HIGH	LOW	HIGH
In2	In4	LOW	LOW	HIGH	HIGH

La MCC commandée par un µC

		Arrêt (moteur libre)	Sens +	Sens –	Arrêt (moteur freiné)
Moteur A	Moteur B				
In1	In3	LOW	HIGH	LOW	HIGH
In2	In4	LOW	LOW	HIGH	HIGH

[Source]

- Les ports **ENA** et **ENB** permettent de gérer l'amplitude de la tension délivrée au moteur, grâce à un signal **PWM** (ou **MLI**).
- Les ports **In1**, **In2** pour le **moteur A** et **In3**, **In4** pour le **moteur B**, permettent de contrôler le sens de rotation des moteurs.

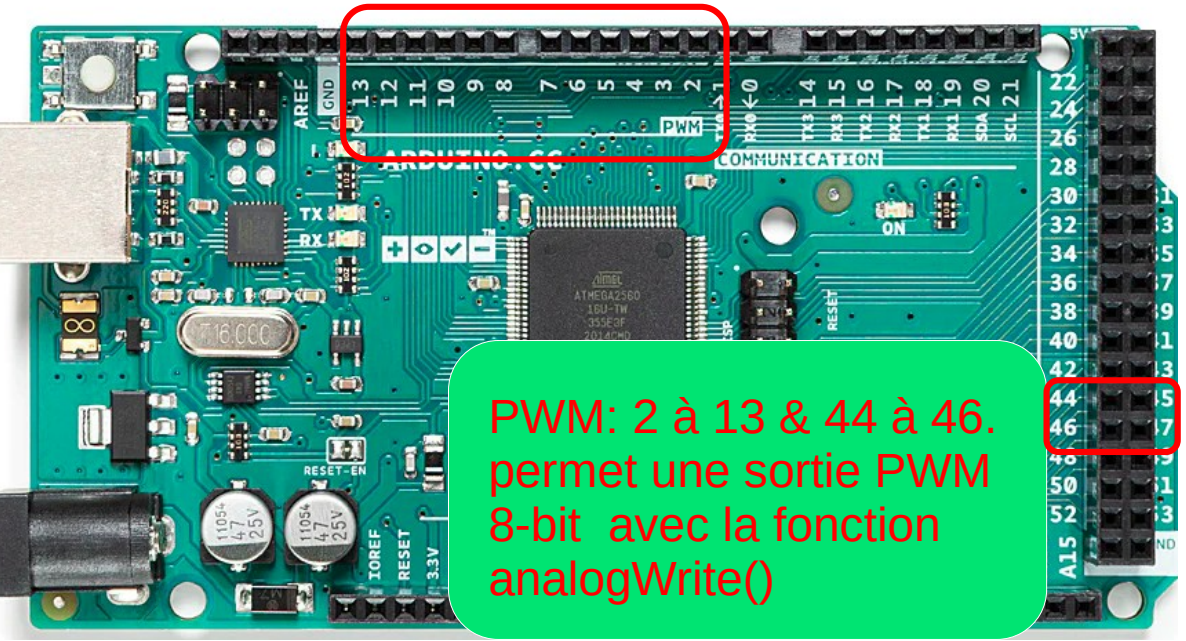
La MCC commandée par un µC

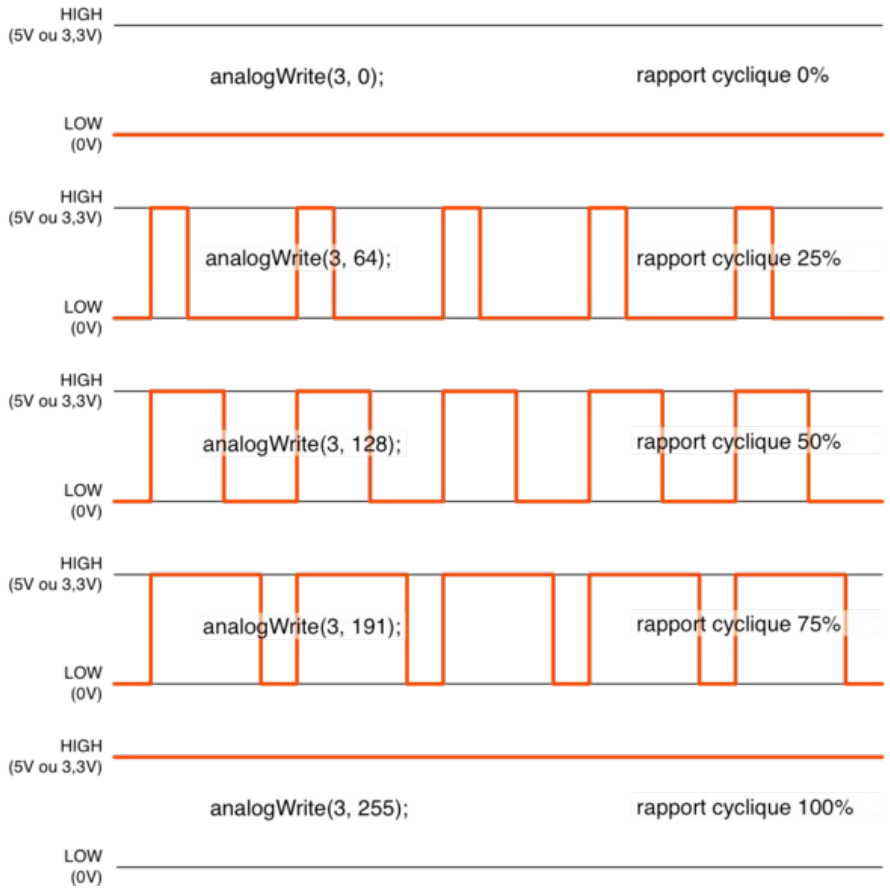
microcontrôleur → L298N → 2 x MCC ou 1 x MCC

La MCC commandée par un µC

microcontrôleur → L298N → 2 x MCC ou 1 x MCC

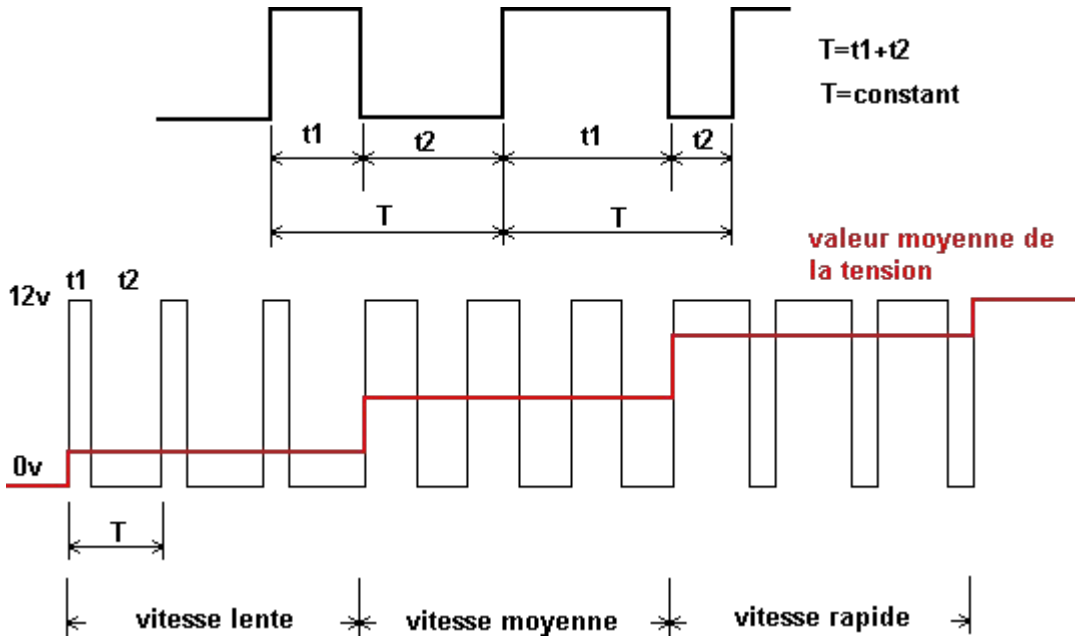
Au niveau logiciel : PWM (MLI)

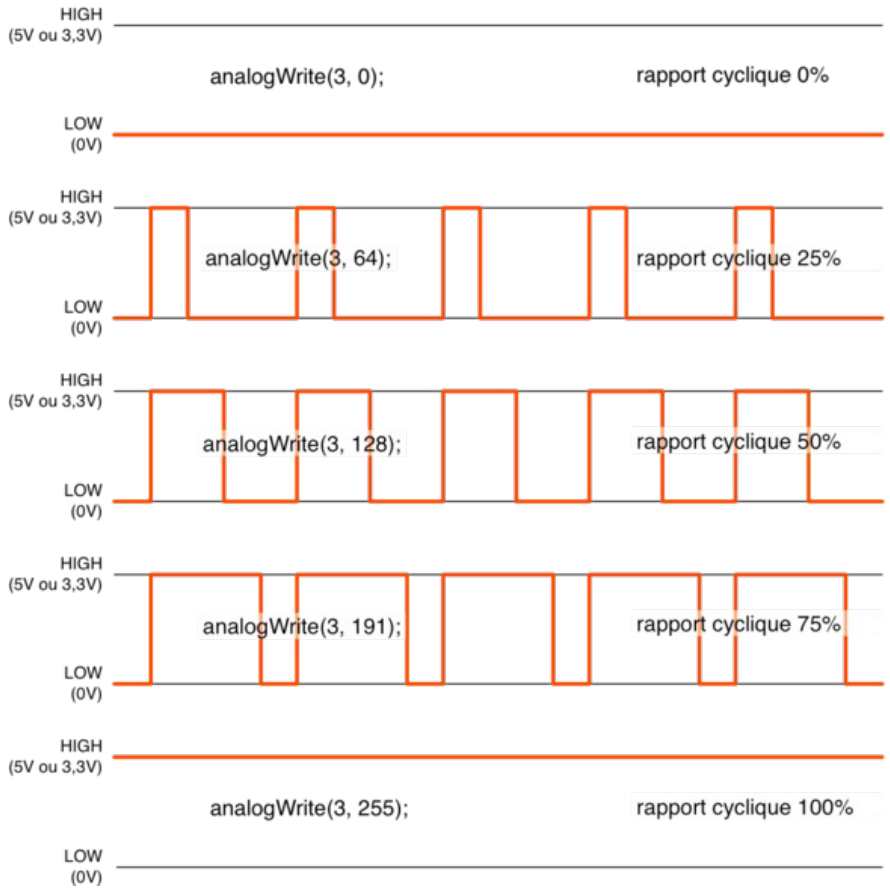




La MCC commandée par un µC

microcontrôleur → L298N → 2 x MCC ou 1 x MCC

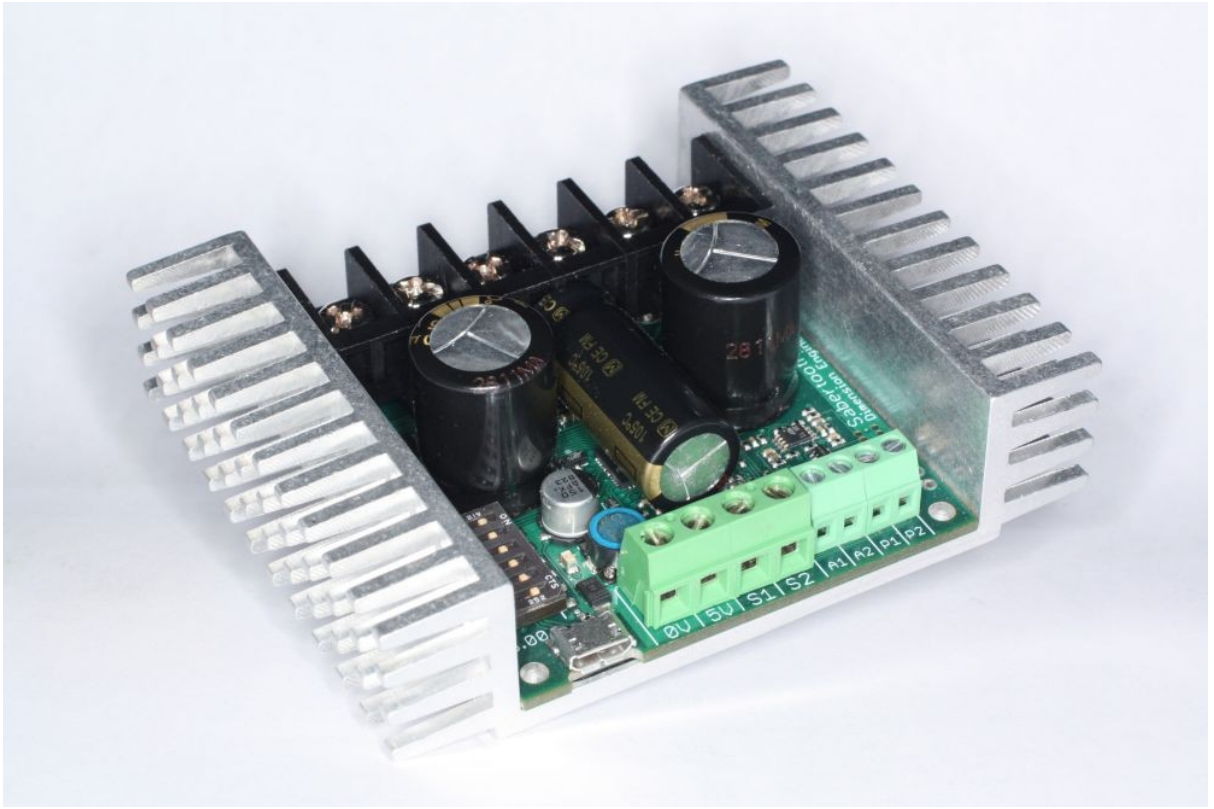




$$V_{out} = V_{moyen} = \frac{1}{T}\int_0^T V_{input}.dt$$

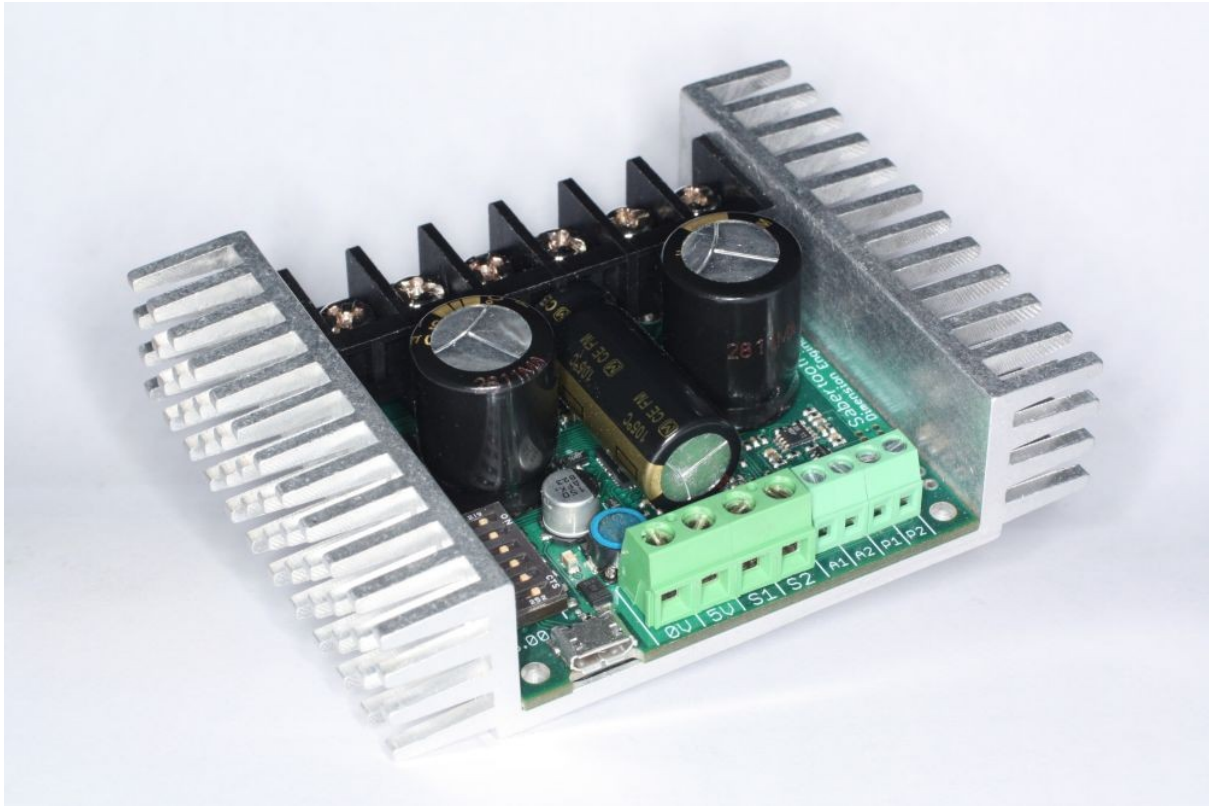
$V_{out} = \alpha.V_{input} = \alpha.12V$

La MCC commandée par un µC



Sabertooth 2x32A

La MCC commandée par un µC

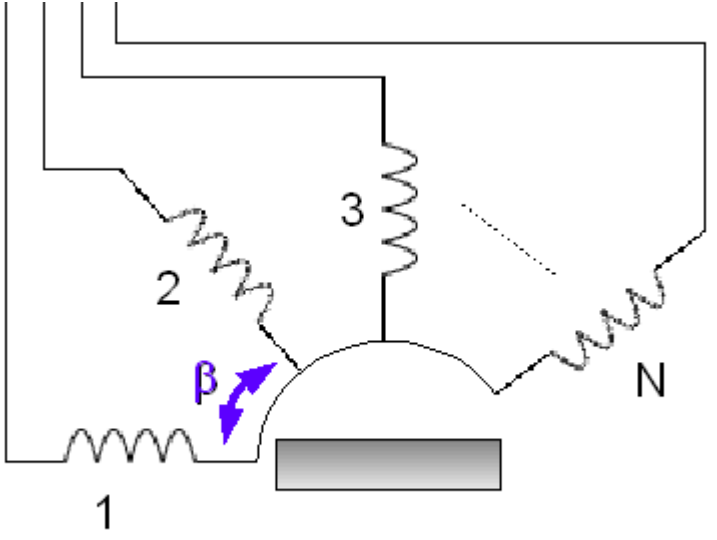


Datasheet
DIP wizard

Sabertooth 2x32A

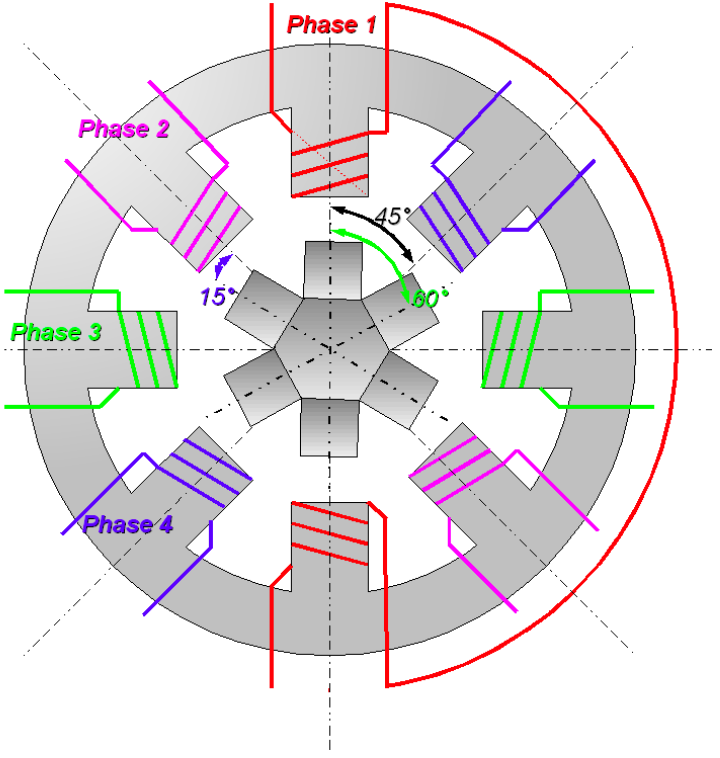
Le MPP
Moteur à Pas à Pas

Le MPP
Moteur à Pas à Pas



Le principe du MPP

Le MPP
Moteur à Pas à Pas



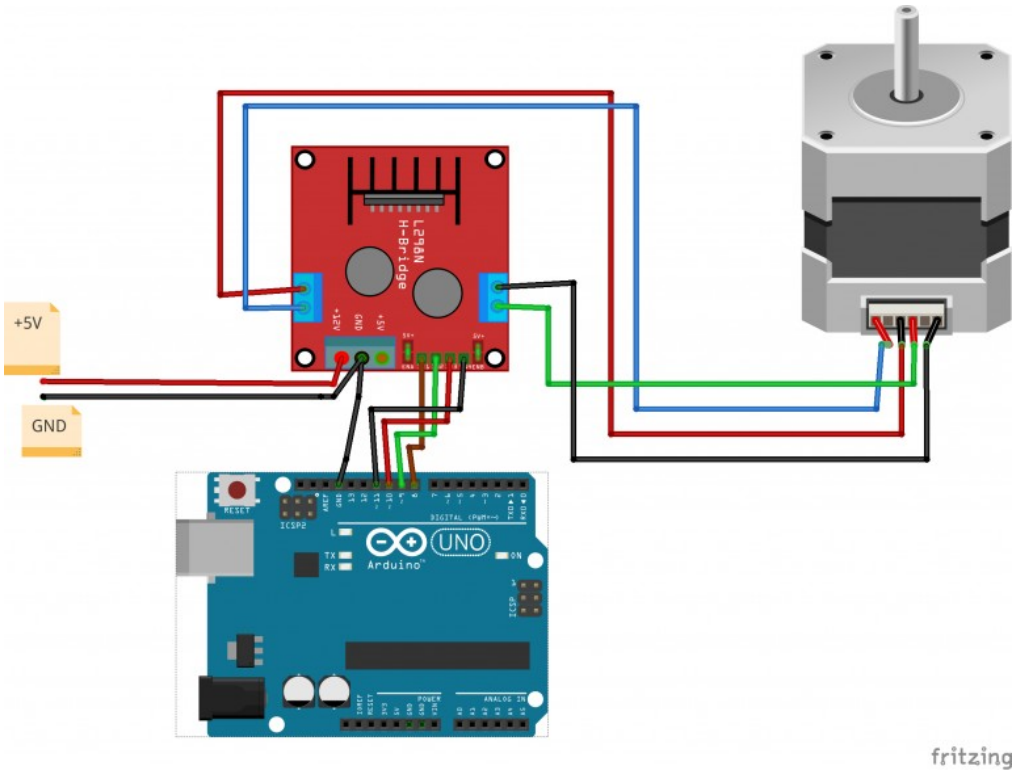
Le MPP
L298 & Moteur à Pas à Pas

Pour moteur pas à pas bipolaire (2 bobines)

L298

Le MPP
L298 & Moteur à Pas à Pas

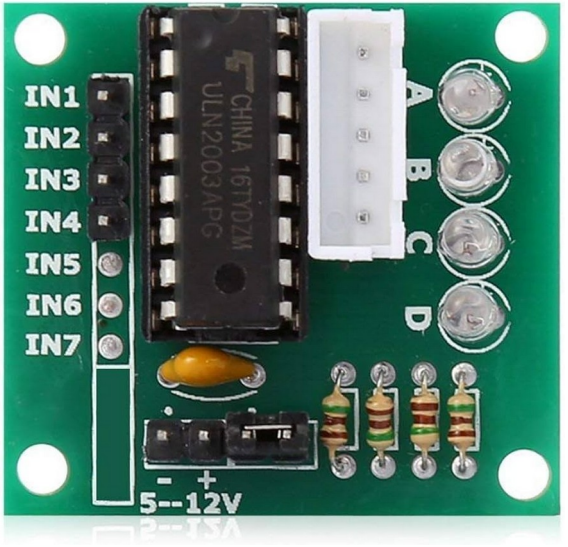
Pour moteur pas à pas bipolaire (2 bobines)



[Source]

Le MPP
ULN2003 & Moteur à Pas à Pas

Pour moteur pas à pas bipolaire (2 bobines)

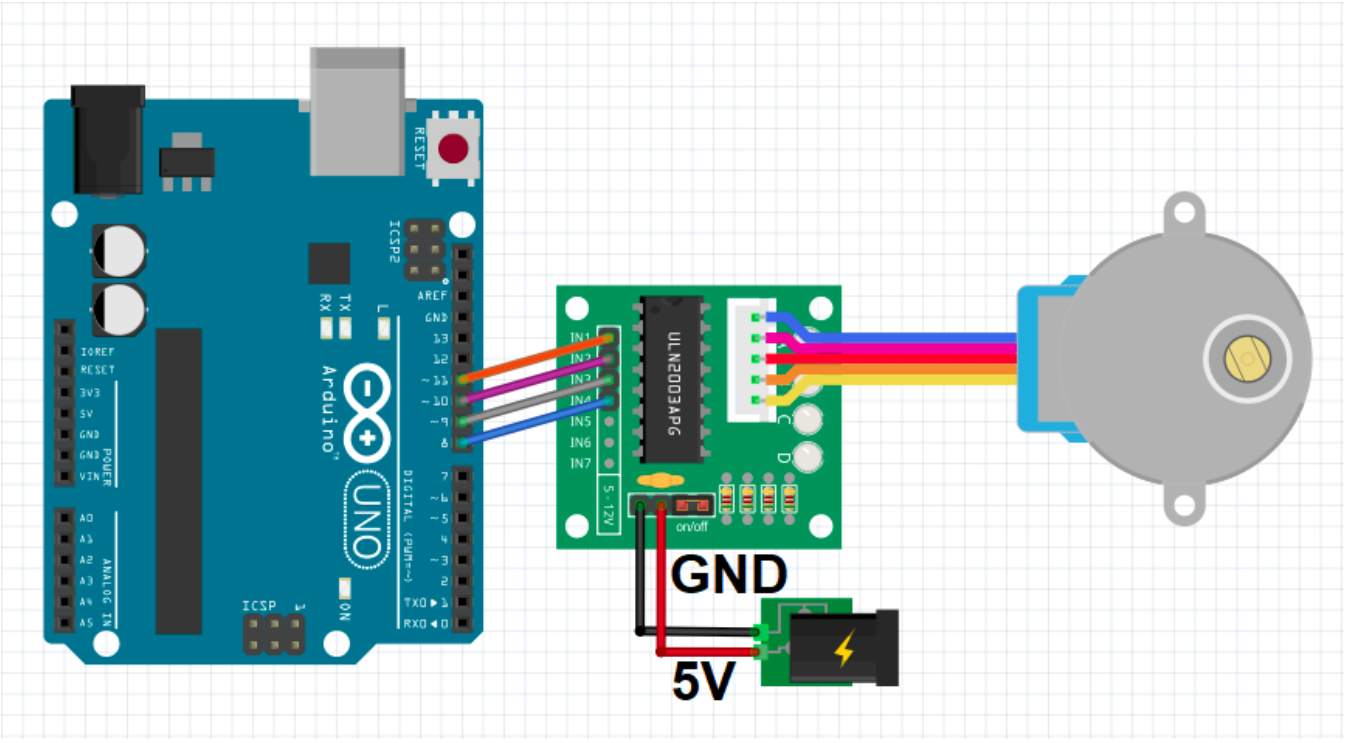


Datasheet du ULN2003

Courant max : 500mA...

Le MPP
ULN2003 & Moteur à Pas à Pas

Pour moteur pas à pas bipolaire (2 bobines)



Le MPP

Activité pratique

TP MCC & MPP

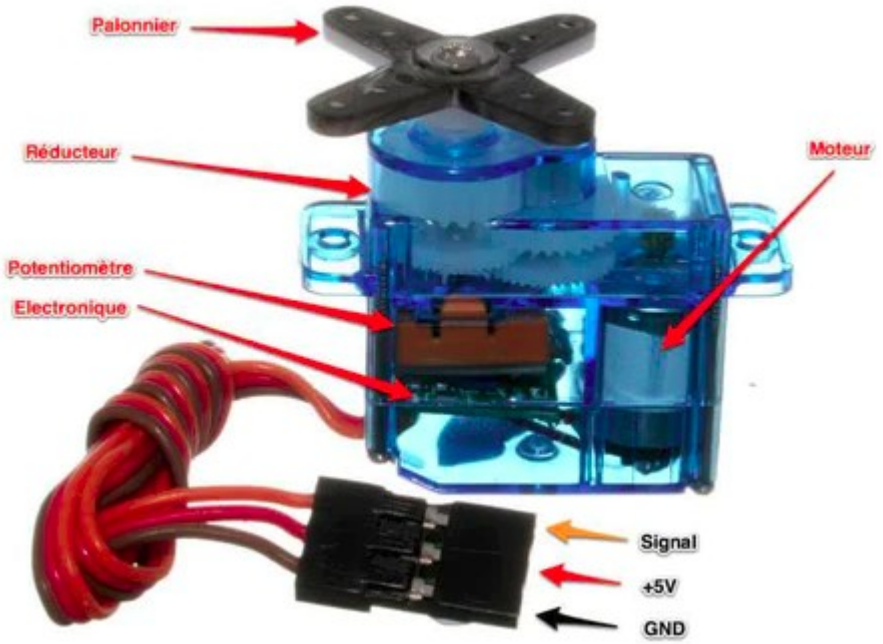
- Mettre en œuvre le câblage & faire vérifier
- Programmer & tester
- Observer à l’oscilloscope et au voltmètre (position DC)
- Modifier les programmes (2 sens de rotation)

Les programmes sont dans le répertoire

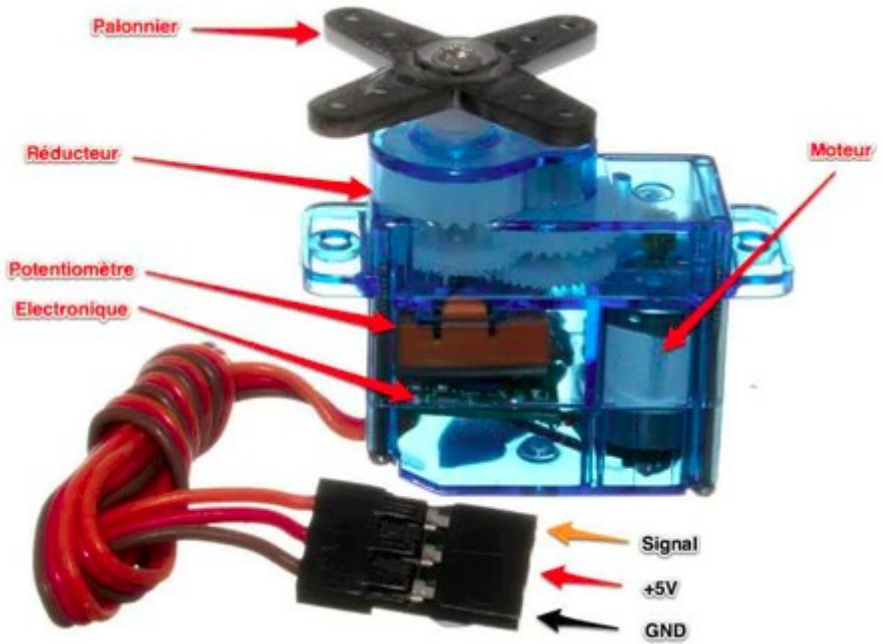
C’est la fin de la 1ère partie !

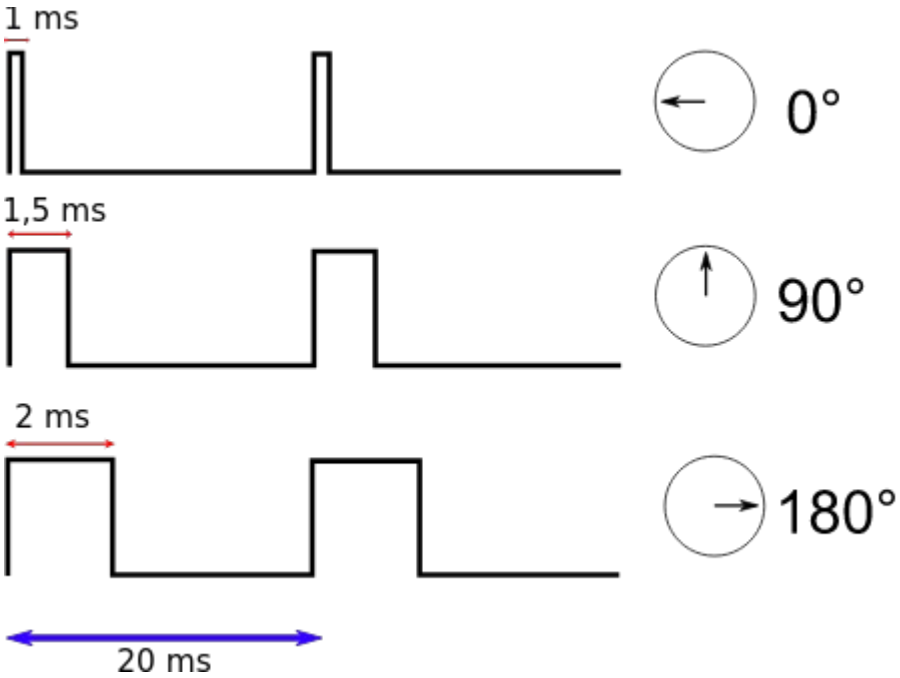
La commande d’angle
Le servomoteur

La commande d’angle
Le servomoteur



La commande d’angle
Le servomoteur





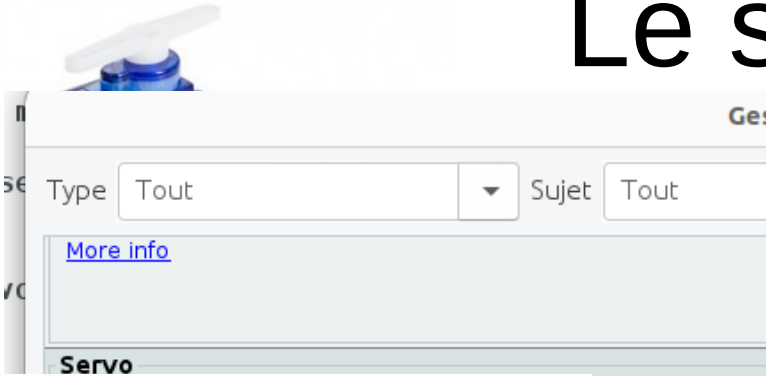
+5V : Rouge
GND : Marron
Signal : Orange

La commande d'angle
Le servomoteur

+5V : Rouge
GND : Marron
Signal : Orange

La com

Le s



- Arduino Micro
- Arduino Leonardo
- Arduino Mega
- Arduino Due
- Arduino MKR FOX 1200
- Arduino MKR GSM 1400
- Arduino MKR NB 1500
- Arduino MKR VIDOR 4000
- Arduino MKR WAN 1300 (LoRa connectivity)
- Arduino MKR WAN 1310
- Arduino MKR WiFi 1010
- Arduino MKR ZERO (I2S bus & SD for sound, music & digital audio data)
- Arduino MKR1000 WIFI
- Arduino Nano
- Arduino Nano 33 BLE
- Arduino Nano 33 IoT
- Arduino Nano Every
- Arduino Uno
- Arduino Uno WiFi REV2
- Arduino Yún
- Arduino Zero
- Portenta H7
- Arduino UNO R4 Minima
- Arduino UNO R4 WiFi
- Arduino Nano RP2040 Connect

Servo

Device Control

Allows Arduino boards to control a variety of servo motors.
This library can control a great number of servos. It makes careful use of timers: the library can control 12 servos using only 1 timer. On the Arduino Due you can control up to 60 servos.

Go to repository

Compatibility

This library is compatible with the **avr, megaavr, sam, samd, nrf52, stm32f4, mbed, mbed_nano, mbed_portenta, mbed_rp2040, renesas, renesas_portenta, renesas_uno** architectures so you should be able to use it on the following Arduino boards:

Usage

This library allows an Arduino board to control RC (hobby) servo motors. Servos have integrated gears and a shaft that can be precisely controlled. Standard servos allow the shaft to be positioned at various angles, usually between 0 and 180 degrees. Continuous rotation servos allow the rotation of the shaft to be set to various speeds.

The Servo library supports up to 12 motors on most Arduino boards and 48 on the Arduino Mega. On boards other than the Mega, use of the library disables `analogWrite()` (PWM) functionality on pins 9 and 10, whether or not there is a Servo on those pins. On the Mega, up to 12 servos can be used without interfering with PWM functionality; use of 12 to 23 motors will disable PWM on pins 11 and 12.

To use this library:

```
#include <Servo.h>
```

Circuit

Servo motors have three wires: power, ground, and signal. The power wire is typically red, and should be connected to the 5V pin on the Arduino board. The ground wire is typically black or brown and should be connected to a ground pin on the Arduino board. The signal pin is typically yellow, orange or white and should be connected to a digital pin on the Arduino board. Note that servos draw considerable power, so if you need to drive more than one or two, you'll probably need to power them from a separate supply (i.e. not the 5V pin on your Arduino). Be sure to connect the grounds of the Arduino and external power supply together.

Servo - writeMicroseconds()

Writes a value in microseconds (us) to the servo, controlling the shaft accordingly. On a standard servo, this will set the angle of the shaft. On standard servos a parameter value of 1000 is fully counter-clockwise, 2000 is fully clockwise, and 1500 is in the middle.

Note that some manufactures do not follow this standard very closely so that servos often respond to values between 700 and 2300. Feel free to increase these endpoints until the servo no longer continues to increase its range. Note however that attempting to drive a servo past its endpoints (often indicated by a growling sound) is a high-current state, and should be avoided.

Continuous-rotation servos will respond to the writeMicrosecond function in an analogous manner to the write function.

Syntax

```
servo.writeMicroseconds(us)
```

Parameters

- *servo*: a variable of type Servo
- *us*: the value of the parameter in microseconds (int)

Example

```
#include <Servo.h>

Servo myservo;

void setup()
{
  myservo.attach(9);
  myservo.writeMicroseconds(1500);  // set servo to mid-point
}

void loop() {}
```

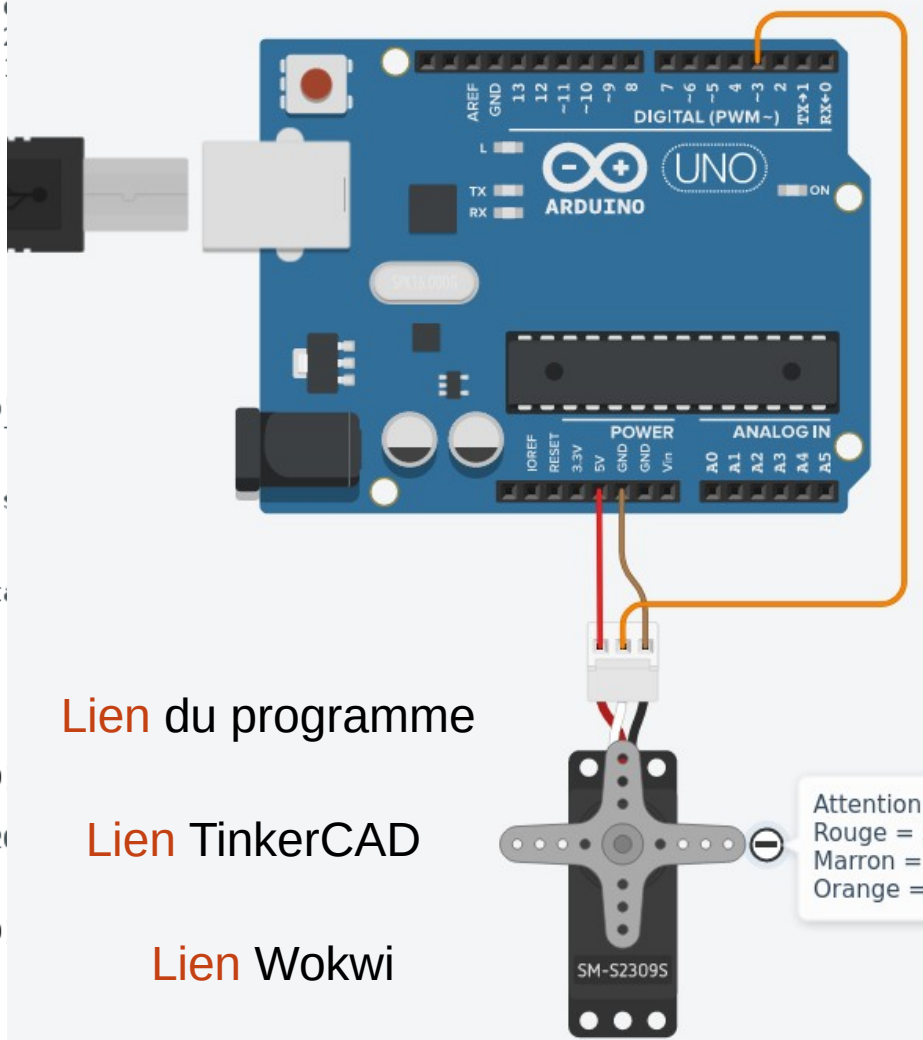
La commande d’angle
Le servomoteur

```
#include <Servo.h>

 const int brocheservo = 3; // broche du signal de commande du servomoteur
 const int maximum=1800; // strictement inférieur à 2000
 const int minimum=1200; // strictement supérieur à 1000

/*
 * 1000 -> 0°
 * 1500 -> 90°
 * 2000 -> 180°
 *
 * => 1° -> incrémentation de 5.55
 * => 20° -> incrémentation de 100
 */

Servo myservo;  // Instanciation de la classe "Servo" en l'objet "myservo"
// plusieurs objets "servo" peuvent être créés pour la plupart des cartes Arduino

int pos = 0;    // variable de position de l'axe du servo

void setup() {
  myservo.attach(brocheservo);  // Le servo est "attaché" à la sortie 3
  myservo.writeMicroseconds(1500);
}

void loop() {
  //myservo.writeMicroseconds(1500);
  for (pos = (minimum); pos <= (maximum); pos += 100) {
    // in steps of 1 degree
    myservo.writeMicroseconds(pos); //varie de 0 à 2000µs
    delay(100);
  }
  for (pos = (maximum); pos >= (minimum); pos -= 100) {
    myservo.writeMicroseconds(pos);
    delay(100);
  }
//myservo.writeMicroseconds(1500);
}
```

```ino
#include <Servo.h>

 const int brocheservo = 3; // broche du signal de commande du servomoteur
 const int maximum=1800; // strictement inférieur à 2000
 const int minimum=1200; // strictement supérieur à 1000

/*
 * 1000 -> 0°
 * 1500 -> 90°
 * 2000 -> 180°
 *
 * => 1° -> incrémentation de 5.55
 * => 20° -> incrémentation de 100
 */

Servo myservo;  // Instanciation de la classe "Servo" en l'objet "myservo"
// plusieurs objets "servo" peuvent être créés pour la plupart des cartes Arduino

int pos = 0;    // variable de position de l'axe du servo

void setup() {
  myservo.attach(brocheservo);  // Le servo est "attaché" à la sortie 3
  myservo.writeMicroseconds(1500);
}

void loop() {
  //myservo.writeMicroseconds(1500);
  for (pos = (minimum); pos <= (maximum); pos += 100) {
    // in steps of 1 degree
    myservo.writeMicroseconds(pos); //varie de 0 à 2000µs
    delay(100);
  }
  for (pos = (maximum); pos >= (minimum); pos -= 100) {
    myservo.writeMicroseconds(pos);
    delay(100);
  }
//myservo.writeMicroseconds(1500);
}
```

Lien du programme

```ino
#include <Servo.h>

 const int brocheservo = 3; // broche du signal de c
 const int maximum=1800; // strictement inférieur à 
 const int minimum=1200; // strictement supérieur à 

/*
 * 1000 -> 0°
 * 1500 -> 90°
 * 2000 -> 180°
 *
 * => 1° -> incrémentation de 5.55
 * => 20° -> incrémentation de 100
 */

Servo myservo;  // Instanciation de la classe "Servo
// plusieurs objets "servo" peuvent être créés pour 

int pos = 0;    // variable de position de l'axe du 

void setup() {
  myservo.attach(brocheservo);  // Le servo est "att
  myservo.writeMicroseconds(1500);
}

void loop() {
  //myservo.writeMicroseconds(1500);
  for (pos = (minimum); pos <= (maximum); pos += 100
    // in steps of 1 degree
    myservo.writeMicroseconds(pos); //varie de 0 à 2
    delay(100);
  }
  for (pos = (maximum); pos >= (minimum); pos -= 100
    myservo.writeMicroseconds(pos);
    delay(100);
  }
//myservo.writeMicroseconds(1500);
}
```



Lien du programme

Lien TinkerCAD

Lien Wokwi

La commande d’angle
Le servomoteur

Activité pratique

Le programme est dans le répertoire

La commande d’angle
L’AX12

La commande d’angle
L’AX12

Repère

Les servomoteurs Dynamixel sont des actuateurs intégrés qui comprennent:

- Un réducteur (!!! plastique !!!)
- Un contrôleur
- Un pilote

Quelques caractéristiques :

- 1024 divisions par tour
- alarme en fonction de la température, du couple ou de la tension d'alimentation
- LED d'état

60€ TTC

La commande d’angle
L’AX12

Repère

Important :

- **Tension d’alimentation : 9 ~ 12V (11.1V recommandé)**
- Angle opérationnel : 0° ~ 300° ou Rotation continue
- Signal de commande : Digital Packet
- Protocole: Half duplex Asynchronous Serial Communication (8 bit, 1 bit de stop, No Parity)
- Lien (Physique) : TTL Level Multi Drop (daisy chain type Connector)
- ID : 254 ID (0~253)
- Vitesse de communication: 7343bps ~ 1 Mbps
- Dimension : 32mm * 50mm * 40mm
- Résolution : 0.29°
- Rapport de réduction : 254 : 1
- Couple de décrochage : 1.52N.m (à 12.0V, 1.5A)
- Vitesse sans charge : 59rpm (at 12V)

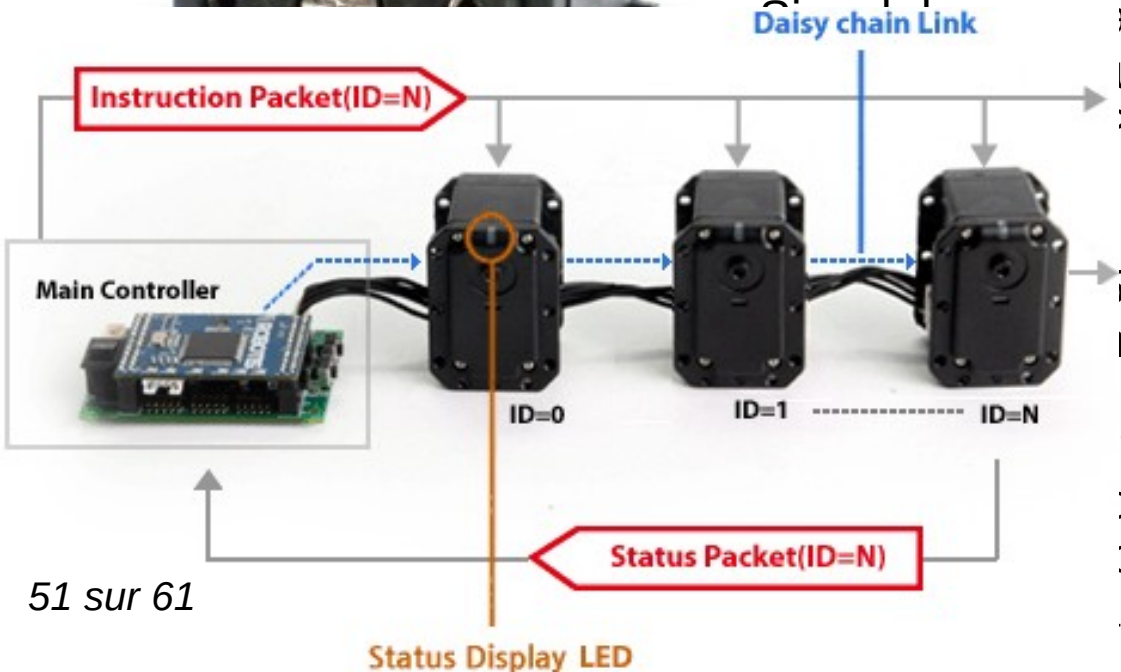
60€ TTC

La commande d'angle
L'AX12

Repère

Important :

- **Tension d'alimentation : 9 ~ 12V (11.1V recommandé)**
- Angle opérationnel : 0° ~ 300° ou Rotation continue
- Si[illegible]
- ande : Digital Packet
- uplex Asynchronous Serial Communication
- arity)
- TTL Level Multi Drop (daisy chain type



- 3)
- unication: 7343bps ~ 1 Mbps
- m * 50mm * 40mm
- °
- tion : 254 : 1
- chage : 1.52N.m (à 12.0V, 1.5A)
- rge : 59rpm (at 12V)

La comma… L’…

Daisy chain (chaîne en guirlande) : 4 fils
pin 1 : GND
Pin 2 : +11,1V
Pin 3 et pin 4 : TX & RX (reliés sur l’Arduino)

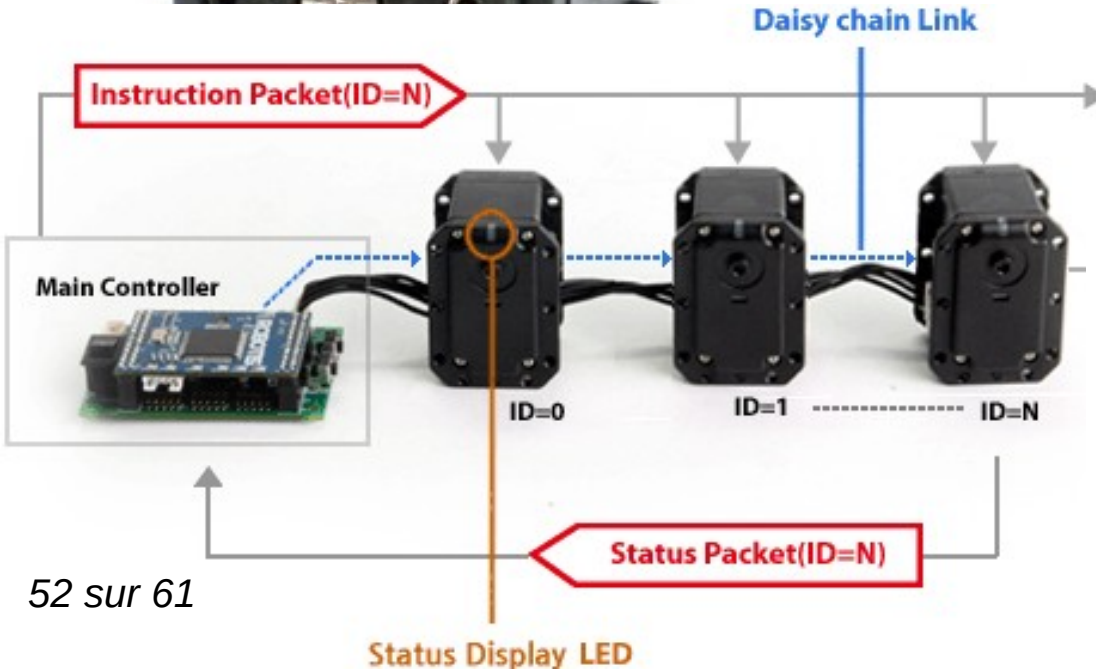
Repère

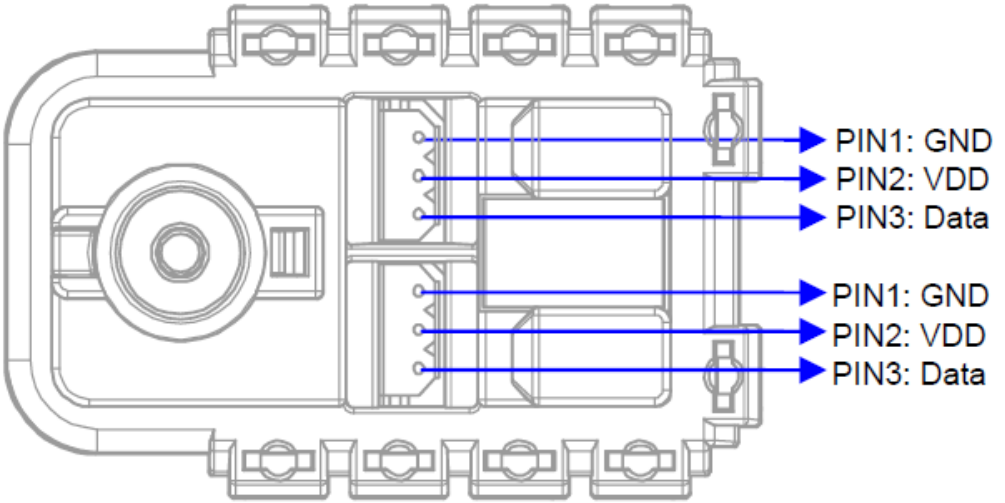
Important :

- **Tension d’alimen… ~ 12V (11.1V recommandé)**

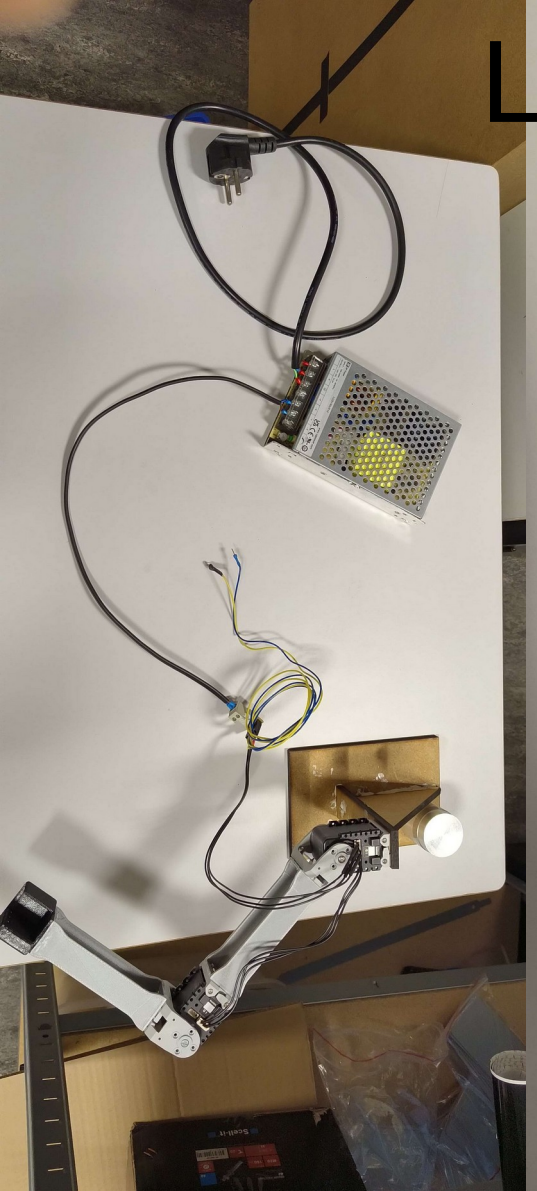
… ~ 300° ou Rotation continue

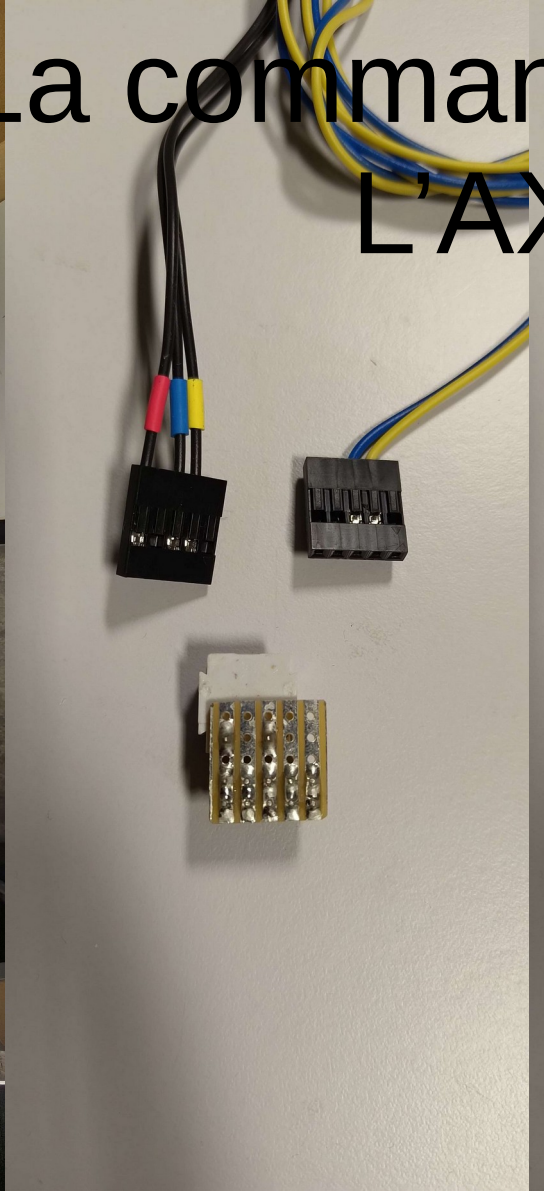
…nde : Digital Packet

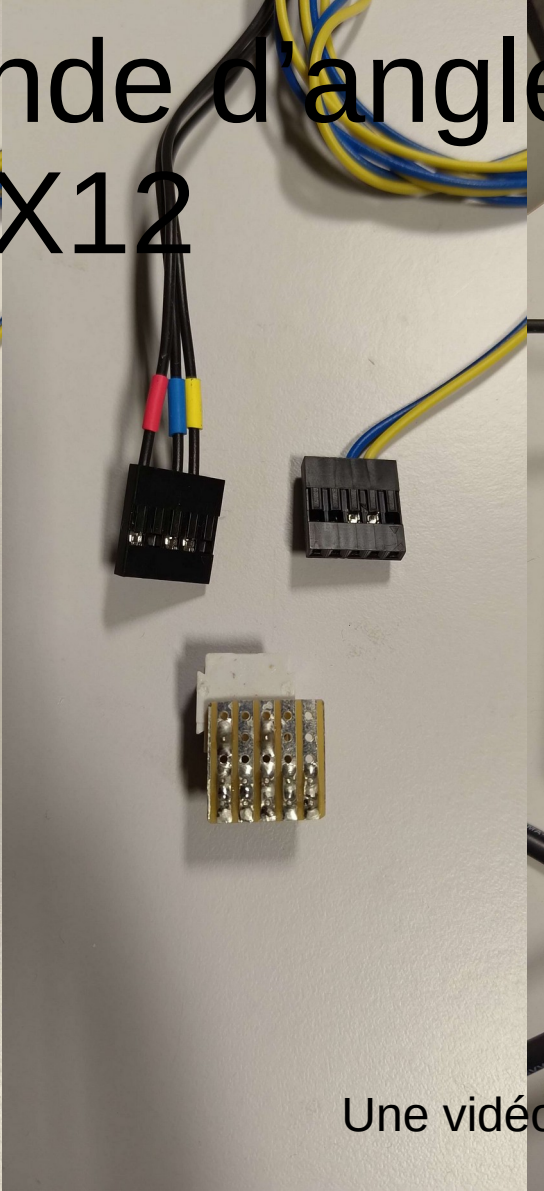


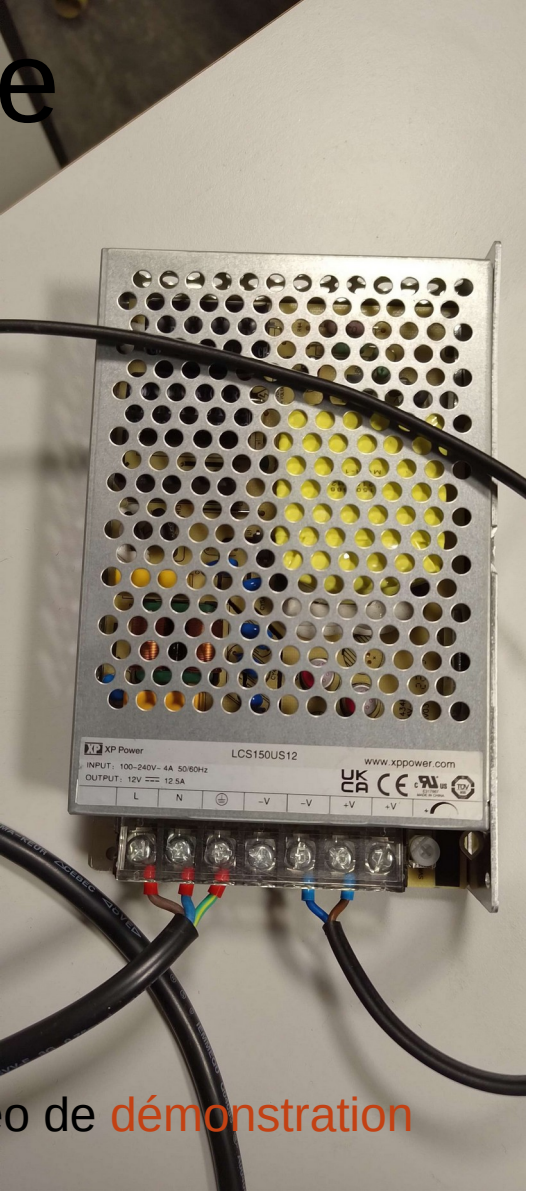


La commande d’angle
L’AX12



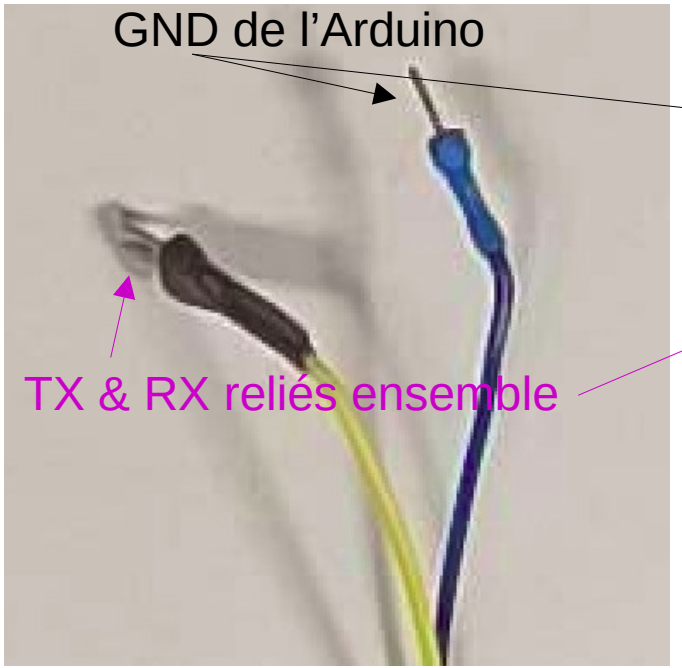


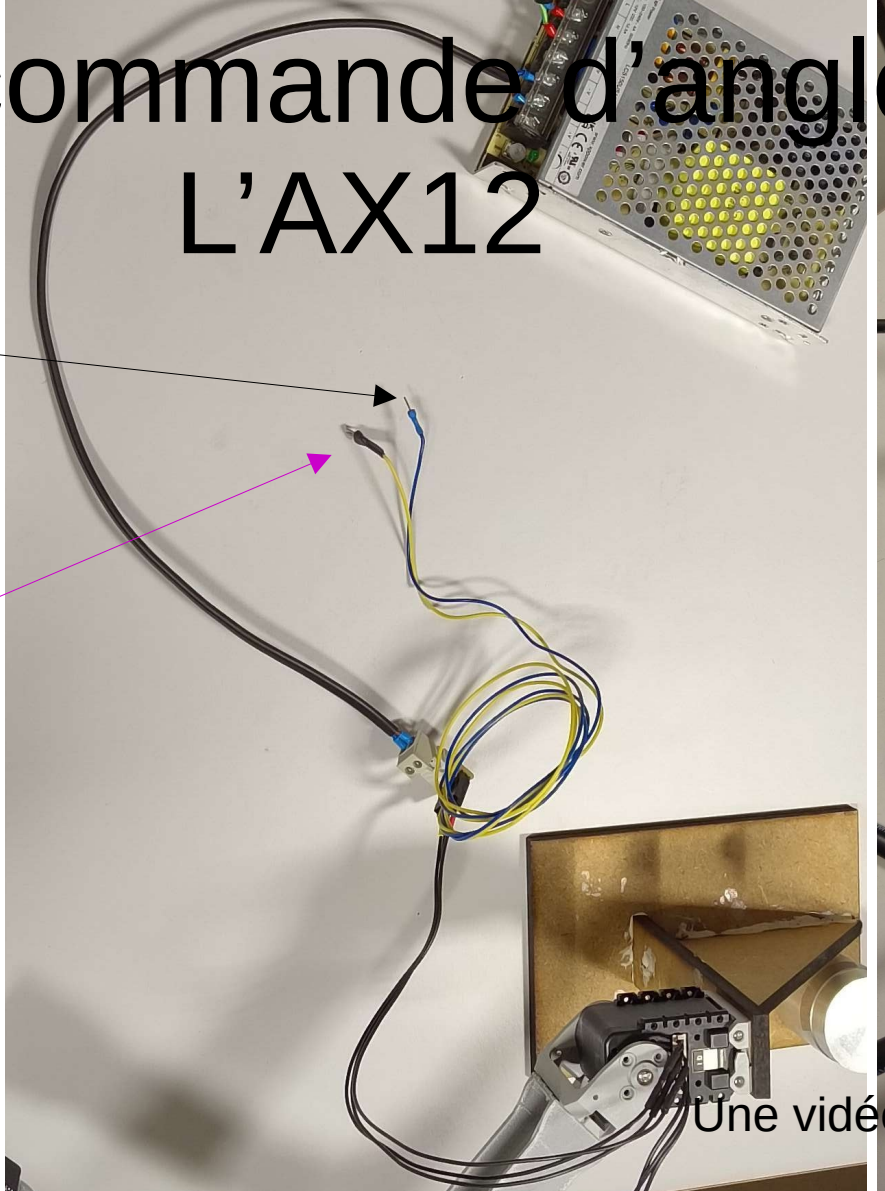


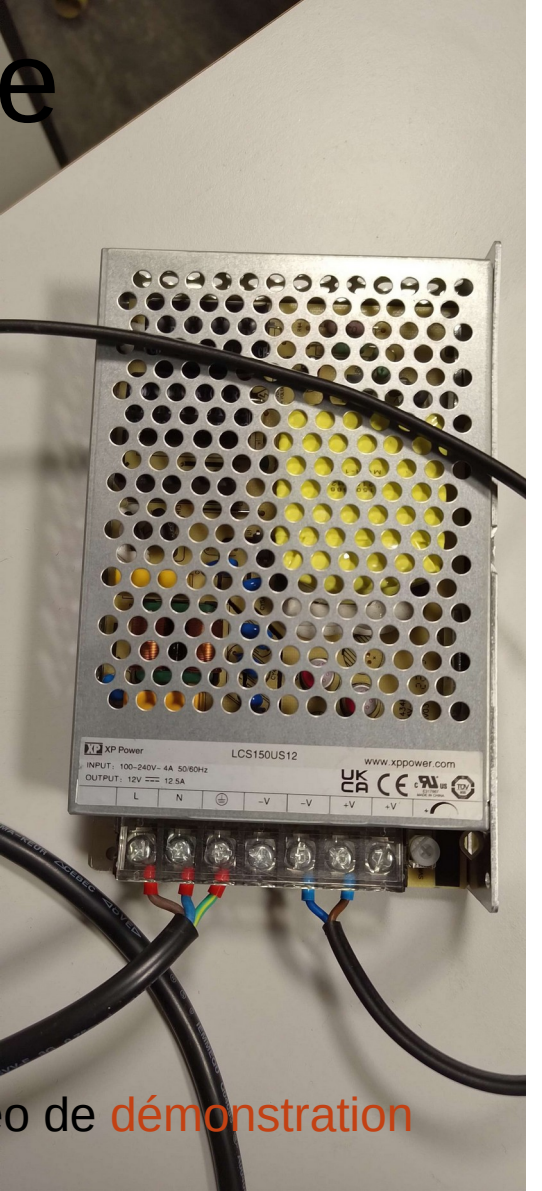


Une vidéo de démonstration

La commande d’angle
L’AX12

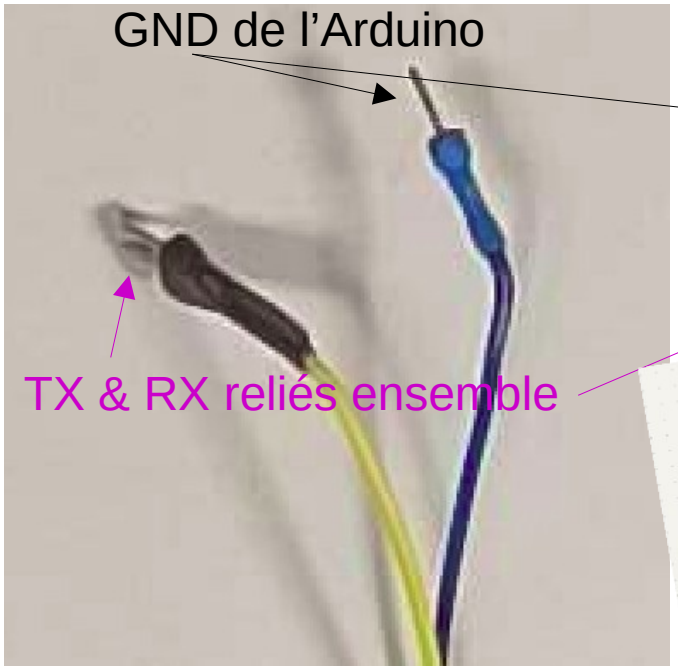


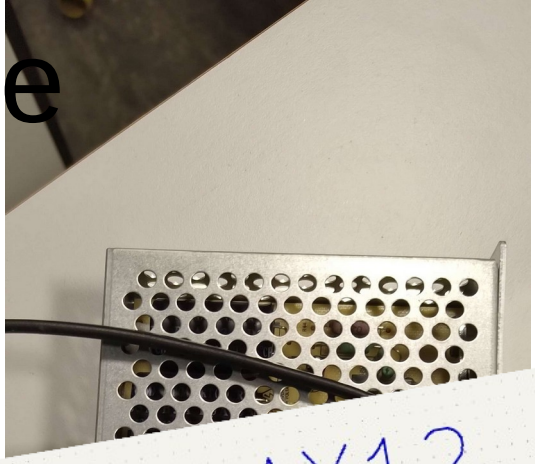


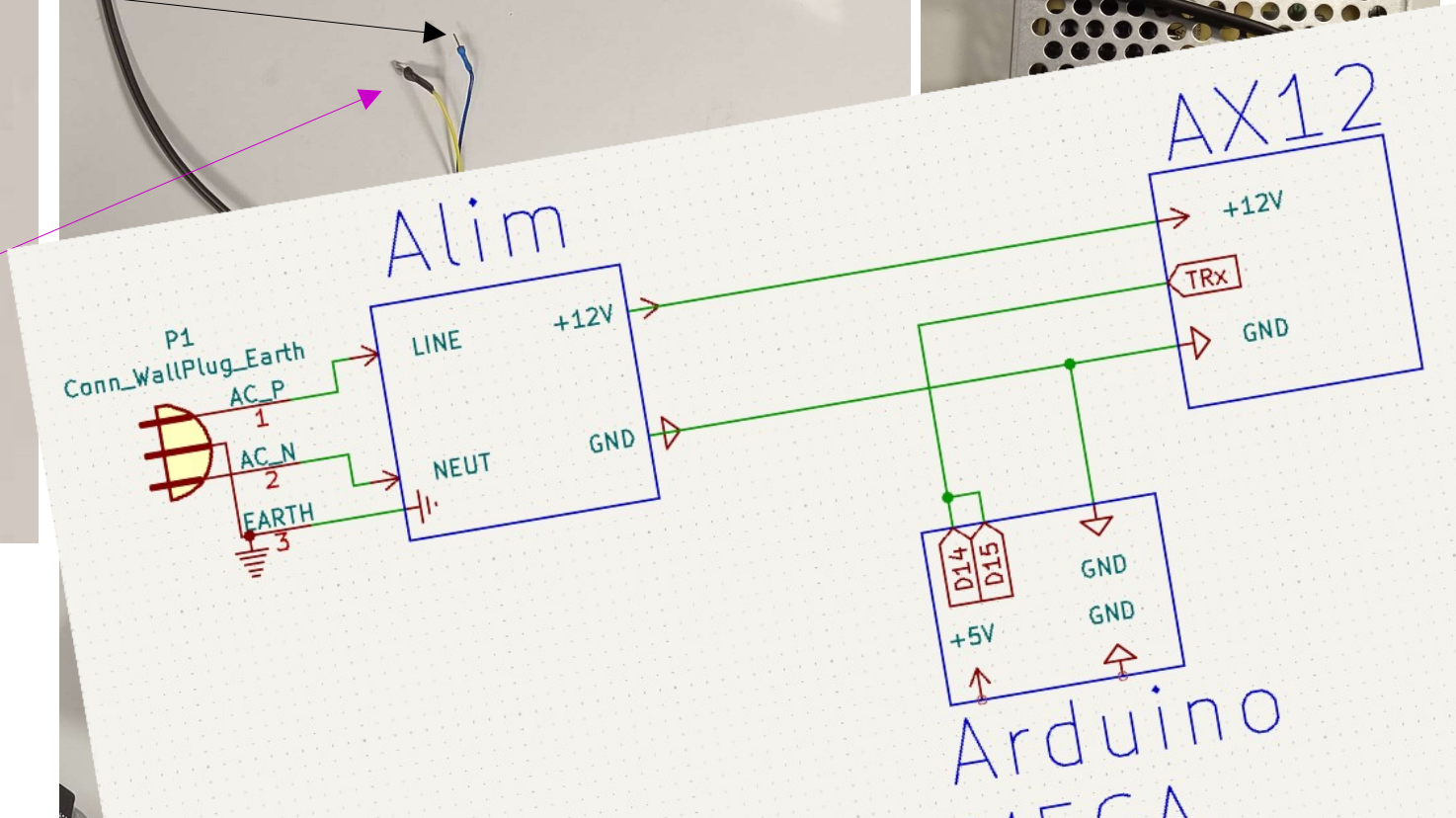


Une vidéo de démonstration

La commande d’angle
L’AX12







La commande d’angle
L’AX12

La librairie de l’AX12 est sur le drive

Une vidéo de démonstration

C’est la fin !

La commande d’angle
Le servomoteur et ROS

La commande d'angle
Le servomoteur et ROS

Classe ros
Méthode NodeHandle
Instanciée en l'objet nh

```
#include <ros.h>
#include <geometry_msgs/Twist.h>
#include <Servo.h>

ros::NodeHandle  nh;

ros::Subscriber<geometry_msgs::Twist> subTwist("/cmd_vel", twistCb );

void setup()
{
.....
  nh.initNode();
  nh.subscribe(subTwist);
}

void loop()
{
  nh.spinOnce();
  delay(50);
}
```

La commande d'angle
Le servomoteur et ROS

Classe ros
Méthode NodeHandle
Instanciée en l'objet nh

```
#include <ros.h>
#include <geometry_msgs/Twist.h>
#include <Servo.h>

ros::NodeHandle  nh;

ros::Subscriber<geometry_msgs::Twist> subTwist("/cmd_vel", twistCb );

void setup()
{
.....
  nh.initNode();
  nh.subscribe(subTwist);
}
void loop()
{
  nh.spinOnce();
  delay(50);
}
```

Classe ros
Méthode Publisher
ros::Publisher pub = node_handle.advertise<message_type>(topic_name, queue_size);
Méthode Subscriber :
ros::Subscriber sub = node_handle.subscribe(topic_name, queue_size, pointer_to_callback_function)

La commande d'angle
Le servomoteur et ROS

```
#include <ros.h>
#include <geometry_msgs/Twist.h>
#include <Servo.h>

ros::NodeHandle  nh;

ros::Subscriber<geometry_msgs::Twist> subTwist("/cmd_vel", twistCb );
```

```
void twistCb( const geometry_msgs::Twist& twist_msg){
  val=twist_msg.angular.z*500+1500;
  if (val<minimum)
  {
    val=minimum;
  }
  {
    val=maximum;
  }
  myservo.writeMicroseconds(val);

  valAV=twist_msg.linear.x*500+1535;
  if (valAV<minimumAV)
  {
    valAV=minimumAV;
  }
  if (valAV>maximumAV)
  {
    valAV=maximumAV;
  }
  myservoAV.writeMicroseconds(valAV);
}
```

Info issues de la manette