Plan de Cours

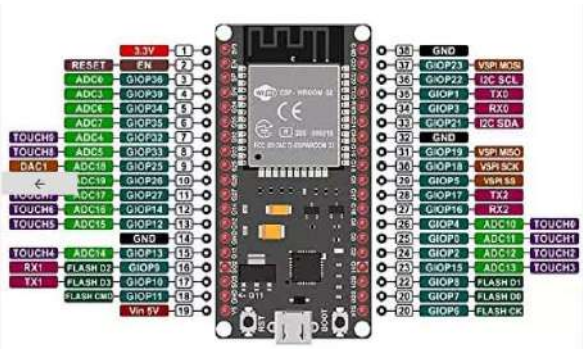
1. Projets réalisés
2. Introduction
3. Le marché de l'IOT
4. Interactions entre le « monde numérique » et le « monde physique »
5. Principes fondamentaux de l'IoT
6. **Composants de base de l'IoT**

- **Node MCU ESP32**
- **Capteurs**
  - **Rôle des capteurs dans l'IoT**
  - **Types de capteurs (température, mouvement, luminosité, etc.)**
- **Actuateurs**
  - **Rôle des actuateurs**
  - **Exemples d'actuateurs (moteurs, relais, etc.)**
- **Raspberry pi 3P**

I. NODE MCU ESP32 (1/1)

ESP32 WIFI

Connectivité sans fil	**-WiFi: 150.0 Mbps débit de données avec HT40** **-Bluetooth: BLE (Bluetooth Low Energy) et Bluetooth Classic**
Processeur	Microprocesseur LX6 double cœur 32 bits, fonctionnant à 160 ou 240 MHz
Mémoire	ROM : 448 KO, SRAM : 520 Ko
Entrée/sortie périphérique	-PWM (modulation de largeur d'impulsion). -ADC (convertisseur analogique-numérique) et DAC -I²C (circuit inter-intégré) -UART (Récepteur/Transmetteur Asynchrone Universel) -SPI (interface périphérique série) -I²S (Interchip Sound intégré) -interface périphérique avec DMA qui inclut le tactile capacitif
Compatible Arduino IDE	vous pouvez programmer l'ESP32 avec l'IDE Arduino (instructions d'installation Windows, Mac OS X et Linux)
Compatible avec MicroPython	vous pouvez programmer l'ESP32 avec le firmware MicroPython (Prise en main de MicroPython sur ESP32)



II. Types de Capteurs (1/10)

Rôle des capteurs dans l'IoT

- Les capteurs collectent des données du monde physique.
- Ils transforment des phénomènes en signaux numériques.
- Les types de capteurs incluent température, mouvement, lumière, gaz, etc.
- Les données des capteurs alimentent les systèmes IoT.
- Les capteurs sont essentiels pour surveiller, contrôler et automatiser des processus.
- Ils sont au cœur de l'Internet des Objets (IoT).

1. **Capteur de Température et d'Humidité DHT22 (ou DHT11)**
2. **Capteur de Mouvement PIR (Passive Infrared)**
3. **Capteur de Luminosité (LDR - Light Dependent Resistor)**
4. **Capteur de Gaz MQ-2**
5. **Capteur d'Humidité du Sol (capteur d'humidité du sol FC-28)**

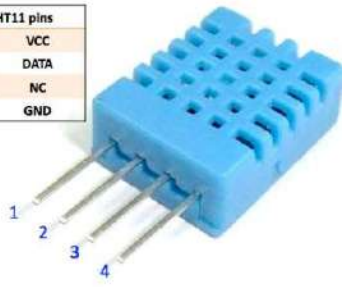
II. Types de Capteurs (1/10)

A. Capteur de Température et d'Humidité DHT22 (ou DHT11)

Mesure la température et l'humidité de l'environnement.

- ❑ Mesure précise de la température (-40°C à 80°C pour DHT11, -40°C à 125°C pour DHT22).
- ❑ Surveillance de l'humidité relative (0% à 100%).
- ❑ Deux versions : DHT22 (AM2302) et DHT11.
- ❑ Sortie numérique pour une intégration facile.
- ❑ Faible consommation d'énergie, idéale pour l'IoT.
- ❑ Applications : régulation du climat, agriculture, domotique.

DHT11 pins	
1	VCC
2	DATA
3	NC
4	GND



II. Types de Capteurs (2/10)

A. Capteur de Température et d'Humidité DHT22 (ou DHT11)

Matériel nécessaire :

- NodeMCU ESP32.
- Capteur de Température et d'Humidité DHT22 (ou DHT11).

Câblage :

- Connectez l'alimentation du capteur à la broche 3.3V du NodeMCU ESP32.
- Connectez la masse du capteur à la broche GND du NodeMCU ESP32.
- Connectez le signal du capteur à une broche GPIO du NodeMCU ESP32 (par exemple, D2).

Code Arduino :

- Vous aurez besoin de la bibliothèque DHT pour lire les données du capteur.

```
#include <DHT.h>
#define DHTPIN D2  // La broche à laquelle le capteur est connecté
#define DHTTYPE DHT22  // Utilisez DHT22 si vous avez le DHT22
DHT dht(DHTPIN, DHTTYPE);
void setup() {
  Serial.begin(115200);
  dht.begin();
}
void loop() {
  delay(2000);  // Attendre quelques secondes entre les mesures
  float humidite = dht.readHumidity();
  float temperature = dht.readTemperature();
  if (isnan(humidite) || isnan(temperature)) {
    Serial.println("Échec de la lecture du capteur !");
  } else {
    Serial.print("Humidité : ");
    Serial.print(humidite);
    Serial.print(" %\t");
    Serial.print("Température : ");
    Serial.print(temperature);
    Serial.println(" °C");
  }
}
```


II. Types de Capteurs (3/10)

B. Capteur de Mouvement PIR (Passive Infrared)

Détecte les mouvements humains en mesurant les changements de chaleur infrarouge.

- ❑ Tension de fonctionnement: DC 5V-20V
- ❑ Consommation électrique statique: 65 microampères
- ❑ Niveau de sortie: haut 3.3V, bas 0V
- ❑ Temps de retard: réglable (0,3 seconde à 10 minutes)
- ❑ Temps de blocage: 0,2 seconde
- ❑ Déclencheur: L ne peut pas être répété, H peut être répété, la valeur par défaut est H
- ❑ Plage d'induction: angle de cône inférieur à 120 degrés, inférieur à 7 mètres
- ❑ Température de travail: -15 -70

II. Types de Capteurs (4/10)

B. Capteur de Mouvement PIR (Passive Infrared)

Matériel nécessaire :

- NodeMCU ESP32.
- Capteur de Mouvement PIR,

Câblage :

- Connectez la broche de signal du capteur PIR à une broche numérique de votre NodeMCU ESP32, par exemple, D2.
- Connectez la masse du capteur à la broche GND du NodeMCU ESP32.
- Connectez la broche d'alimentation (VCC) du capteur PIR à une broche 3.3V du NodeMCU ESP32.

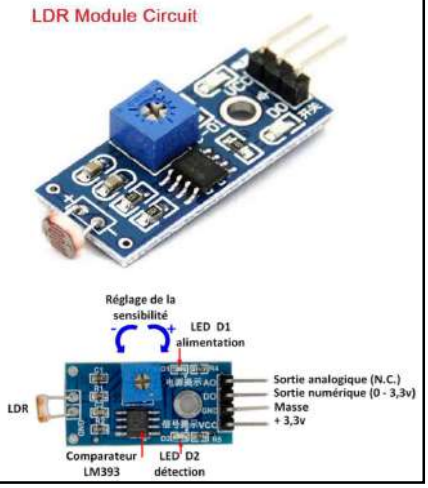
```
#define PIR_PIN D2 // Broche de signal du capteur PIR

void setup() {
  Serial.begin(115200);
  pinMode(PIR_PIN, INPUT);
   // Configure la broche du capteur PIR comme une entrée
}

void loop() {
  int motionState = digitalRead(PIR_PIN);
  // Lis l'état du capteur PIR (HAUT ou BAS)

  if (motionState == HIGH) {
    Serial.println("Mouvement détecté !");
    // Ajoutez ici les actions à effectuer en cas de détection de mouvement
  } else {
    Serial.println("Pas de mouvement.");
    // Ajoutez ici les actions à effectuer lorsque le mouvement s'arrête
  }
  delay(1000);
  // Attendez quelques secondes entre les lectures pour éviter
  //les détections multiples
}
```


II. Types de Capteurs (5/10)

C. Capteur de Luminosité (LDR - Light Dependent Resistor)

Détecte Mesure la luminosité ambiante.

- ❑ Puissance maximale: 100mW
- ❑ Température de fonctionnement: -30 ° C ~ 70 ° C
- ❑ Pic spectral: 540nm
- ❑ Résistance lumineuse (10Lux) (KΩ): 10 - 20
- ❑ Résistance foncée: 1 MΩ
- ❑ 100λ10: 0,6
- ❑ Temps de réponse: 20ms (Rise), 30ms (Down)
- ❑ Éclairage de résistance: 3



II. Types de Capteurs (6/10)

C. Capteur de Luminosité (LDR - Light Dependent Resistor)

```
const int ldrPin = D2; // Broche numérique pour le capteur LDR

void setup() {
  Serial.begin(115200);
  pinMode(ldrPin, INPUT);
}

void loop() {
  int luminosite = digitalRead(ldrPin); // Lit l'état de la broche
  Serial.print("Luminosité : ");
  Serial.println(luminosite);

  delay(1000); // Attendez quelques secondes entre les lectures
}
```

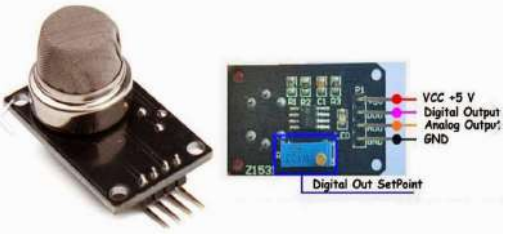
```
#define LDR_PIN A0 // Broche analogique pour le capteur LDR

void setup() {
  Serial.begin(115200);
}

void loop() {
  int luminosite = analogRead(LDR_PIN);
  // Lit la valeur analogique du capteur LDR
  Serial.print("Luminosité : ");
  Serial.println(luminosite);

  delay(1000); // Attendez quelques secondes entre les lectures
}
```


II. Types de Capteurs (7/10)

D. Capteur de Gaz MQ-2

Détecte divers gaz, y compris le méthane, le butane, le propane, le monoxyde de carbone et l'alcool.

- ❑ Alimentation: 5 Vcc
- ❑ Plage de détection: 100-10000 ppm
- ❑ Sortie analogique et numérique
- ❑ Température de fonctionnement : -10 °C à +50 °C



La tension de sortie analogique du module MQ-2 varie en fonction de la concentration de fumée ou de gaz. Plus la concentration de gaz est élevée, plus la tension de sortie est élevée. Le signal logique peut être calibré en tenant le capteur à proximité de la fumée que vous souhaitez détecter.

II. Types de Capteurs (8/10)

D. Capteur de Gaz MQ-2

Matériel nécessaire :

- NodeMCU ESP32.
- Capteur de gaz MQ-2
- Résistance de charge 10K ohms
- Câbles de raccordement
- Alimentation 5V

Câblage :

- Connectez la broche A0 (Analogique 0) du capteur MQ-2 à une broche analogique de l'ESP32, par exemple, A0.
- Connectez la broche D0 (Digital 0) du capteur MQ-2 à une broche numérique de l'ESP32, par exemple, D2.
- Connectez une extrémité de la résistance de charge (10K ohms) à la broche A0 du capteur MQ-2.
- Connectez l'autre extrémité de la résistance de charge au 5V de l'ESP32.
- Connectez la broche GND du capteur MQ-2 au GND de l'ESP32.
- Alimentez le capteur MQ-2 avec une alimentation externe de 5V.

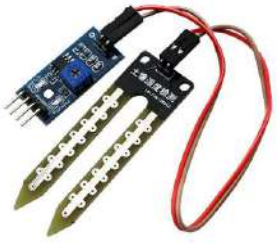
```
const int analogPin = A0; // Broche analogique pour le capteur MQ-2
const int digitalPin = D2; // Broche numérique pour le capteur MQ-2

void setup() {
  Serial.begin(115200);
  pinMode(digitalPin, INPUT);
}

void loop() {
  int value = analogRead(analogPin);
  // Lit la valeur analogique du capteur MQ-2
  int digitalValue = digitalRead(digitalPin);
  // Lit la valeur numérique du capteur MQ-2

  Serial.print("Valeur analogique : ");
  Serial.println(value);
  Serial.print("Valeur numérique : ");
  Serial.println(digitalValue);

  delay(1000); // Attendez quelques secondes entre les lectures
}
```


II. Types de Capteurs (9/10)

E. Capteur d'Humidité du Sol (capteur d'humidité du sol FC-28)

Mesure l'humidité du sol, idéal pour les projets de jardinage automatisé.

- ❑ Tension de fonctionnement: cc 3.3-5.5V
- ❑ Tension de sortie: cc 0-3.0V
- ❑ Interface: PH2.0-3P
- ❑ Taille: 99x1 6mm/3.9x0.63"



II. Types de Capteurs (10/10)

E. Capteur d'Humidité du Sol (capteur d'humidité du sol FC-28)

Matériel nécessaire :

- ESP32 (NodeMCU ESP32 ou équivalent)
- Capteur d'humidité du sol FC-28
- Alimentation (3.3V ou 5V)

Câblage :

- Connectez le module d'interface au capteur FC-28.
- Connectez la broche AO (Analog Out) du module d'interface à une broche analogique de l'ESP32, par exemple, A0.
- Connectez la broche GND du module d'interface au GND de l'ESP32.
- Alimentez le module d'interface avec une alimentation externe de 3.3V ou 5V.

```
const int analogPin = A0;
// Broche analogique pour le capteur FC-28

void setup() {
  Serial.begin(115200);
}

void loop() {
  int value = analogRead(analogPin);
  // Lit la valeur analogique du capteur FC-28

  Serial.print("Niveau d'humidité du sol : ");
  Serial.println(value);

  delay(1000);
  // Attendez quelques secondes entre les lectures
}
```


III. Actuateurs (1/18)

Rôle des Actuateurs dans l'IoT

- Les actionneurs sont des éléments clés de l'IoT.
- Ils transforment les données en actions physiques.
- Exemples : moteurs, relais, électrovannes, lumières, buzzeurs.
- Ils automatisent des tâches, améliorent la sécurité, la gestion de l'énergie, et l'efficacité des systèmes IoT.
- Les actionneurs sont les exécuteurs des décisions IoT dans le monde réel.

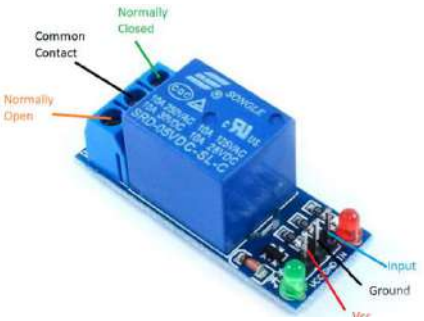
1. **Relais (Relay Module);**
2. **Électrovannes;**
3. **Écrans LCD ou OLED;**
4. **Haut-parleurs ou Buzzer;**
5. **Moteurs DC (DC Motors) ;**
6. **Moteurs pas à pas (Stepper Motors);**
7. **Servomoteurs (Servo Motors).**

IV. Actuateurs (2/18)

A. Relais (Relay Module)

Utilisé pour activer ou désactiver des appareils électriques tels que lumières, ventilateurs, etc.

- **Nombre de Voies** : Un ou plusieurs canaux pour contrôler plusieurs charges.
- **Tension de Commutation** : Fonctionne généralement sous 5 V pour le signal de commande.
- **Courant de Commutation** : Capacité de commutation, par exemple, 10 A pour des charges jusqu'à 10 ampères.
- **Type de Charge** : Compatible avec courant continu (CC) et alternatif (CA).
- **Protection Intégrée** : Diodes de Flyback pour prévenir les pics de tension.
- **Interface** : Broches ou connecteurs standard pour la connexion à l'Arduino.
- **LED d'Indication** : Indique l'état du relais (activé/désactivé).



IV. Actuateurs (3/18)

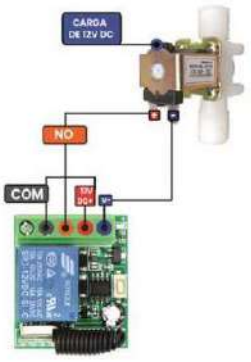
B. Électrovannes

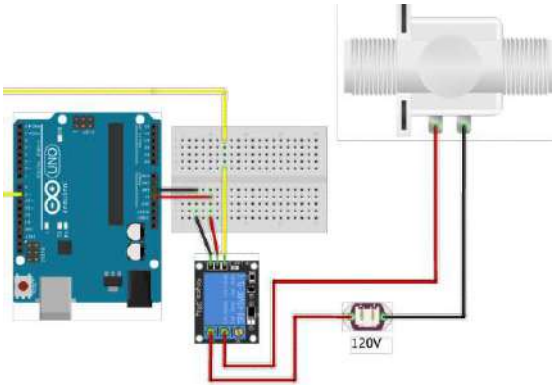
Les électrovannes sont couramment utilisées pour contrôler le flux de liquides ou de gaz, que ce soit pour l'irrigation automatique, la distribution de liquides, les systèmes de contrôle de gaz, etc.

- **Type d'Actionneur** : Électrovanne à actionnement électrique.
- **Tension de Fonctionnement** : Habituellement 12 V ou 24 V, selon le modèle.
- **Type de Contrôle** : Généralement commandé par un signal PWM (modulation de largeur d'impulsion).
- **Débit** : Spécifie le volume d'écoulement de fluide (en litres par minute).
- **Type de Fluide** : Conçu pour l'eau, l'air, les gaz, etc., en fonction de l'application.
- **Type de Connexion** : Filetage mâle ou femelle pour le raccordement aux tuyaux.

IV. Actuateurs (4/18)

B. Électrovannes





IV. Actuateurs (5/18)

B. Électrovannes

```
#include <Arduino.h>
// Broches de connexion
const int boutonPin = 2;  // Broche du bouton, à adapter à votre configuration
const int relaisPin = 4;  // Broche de contrôle du relais, à adapter à votre configuration
// Variables de statut du relais et du bouton
bool etatRelais = LOW; // LOW = éteint, HIGH = allumé
bool etatBoutonPrecedent = LOW;
bool etatBoutonActuel = LOW;
void setup() {
  pinMode(boutonPin, INPUT_PULLUP); // Configuration du bouton avec résistance de pull-up interne
  pinMode(relaisPin, OUTPUT);
  digitalWrite(relaisPin, etatRelais);
  Serial.begin(115200);
}
void loop() {
  // Lecture de l'état actuel du bouton
  etatBoutonActuel = digitalRead(boutonPin);

  // Vérification du changement d'état du bouton
  if (etatBoutonActuel != etatBoutonPrecedent) {
    delay(50); // Anti-rebond
    etatBoutonActuel = digitalRead(boutonPin);

    // Si le bouton est enfoncé, basculez l'état du relais
    if (etatBoutonActuel == LOW) {
      etatRelais = !etatRelais;
      digitalWrite(relaisPin, etatRelais);
    }
  }
  etatBoutonPrecedent = etatBoutonActuel;
}
```

IV. Actuateurs (6/18)

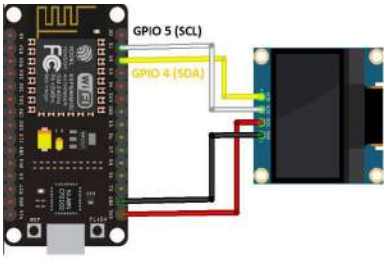
C. Écrans LCD ou OLED

Afficher des informations visuelles, comme les écrans OLED dans les montres intelligentes.

- **Type d'affichage** : Les écrans LCD pour Arduino sont généralement de type alphanumérique, graphique ou TFT (Thin-Film Transistor).
- **Taille de l'écran** : Ils sont disponibles en différentes tailles, de petits écrans 16x2 caractères aux grands écrans TFT couleur.
- **Résolution** : La résolution dépend du type d'écran, avec des écrans alphanumériques offrant généralement une résolution standard.
- **Interface** : Les écrans LCD utilisent généralement une interface parallèle ou série, avec des broches d'entrée/sortie pour la communication avec Arduino.
- **Tension d'alimentation** : Ils fonctionnent généralement avec une tension de 5V ou 3.3V, selon le modèle.
- **Bibliothèques disponibles** : Il existe des bibliothèques Arduino prêtes à l'emploi pour faciliter la programmation des écrans LCD.

IV. Actuateurs (7/18)

C. Écrans LCD ou OLED




```
#include <Wire.h>  // Inclure la bibliothèque Wire pour la communication I2C
#include <LiquidCrystal_I2C.h>  // Inclure la bibliothèque LiquidCrystal_I2C

// Initialisation de l'écran LCD avec l'adresse I2C 0x27 (vous pouvez modifier cette adresse si elle est différente)
LiquidCrystal_I2C lcd(0x27, 16, 2);  // Utilisez 16 colonnes x 2 lignes

void setup() {
  // Initialisation de l'écran LCD
  lcd.init();

  // Allumer le rétroéclairage (peut être supprimé si l'écran s'allume automatiquement)
  lcd.backlight();

  // Afficher un message sur l'écran LCD
  lcd.setCursor(0, 0);  // Position du curseur (colonne, ligne)
  lcd.print("Bonjour, ESP32!");
}

void loop() {
  // Votre code principal peut aller ici
}
```

IV. Actuateurs (8/18)

D. Haut-parleurs ou Buzzer

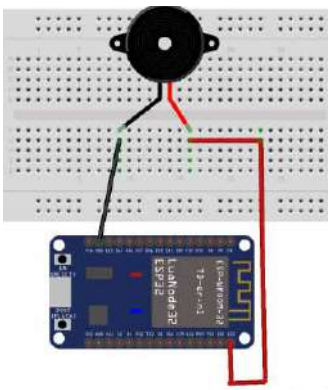
Produire des sons ou des alertes sonores dans les projets audio.

- **Tension de Fonctionnement** : 3,3 V ou 5 V
- **Type de Signal d'Entrée** : Numérique
- **Niveau Sonore** : Variable (mesuré en dB)
- **Fréquence de Résonance** : Spécifique au modèle
- **Type de Contrôle** : Actif ou Passif
- **Type de Connexion** : Diverses options disponibles
- **Taille et Forme** : Variées
- **Compatibilité avec Arduino** : Vérifier la tension et le contrôle

IV. Actuateurs (9/18)

D. Haut-parleurs ou Buzzer

Produire des sons ou des alertes sonores dans les projets audio.



```
// Broche de contrôle du buzzer
int buzzerPin = 13; // Vous pouvez choisir une autre broche

void setup() {
  pinMode(buzzerPin, OUTPUT);
}

void loop() {
  // Jouer la note DO (C)
  tone(buzzerPin, 261.63); // Fréquence pour la note DO
  delay(500);
  noTone(buzzerPin);
  delay(100);

  // Jouer la note MI (E)
  tone(buzzerPin, 329.63); // Fréquence pour la note MI
  delay(500);
  noTone(buzzerPin);
  delay(100);

  // Jouer la note SOL (G)
  tone(buzzerPin, 392.00); // Fréquence pour la note SOL
  delay(500);
  noTone(buzzerPin);
  delay(100);
}
```

IV. Actuateurs (10/18)

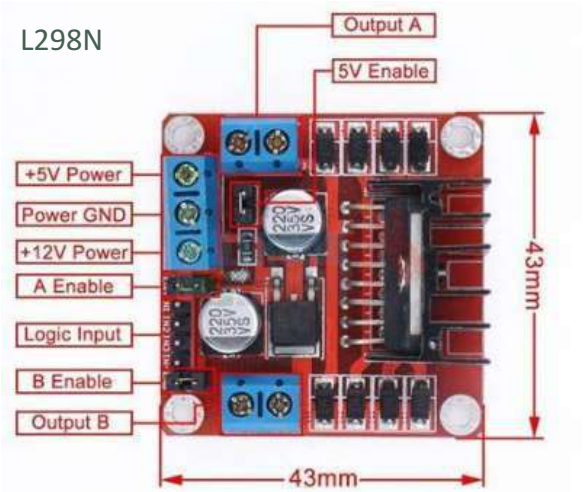
E. Moteurs DC (DC Motors)

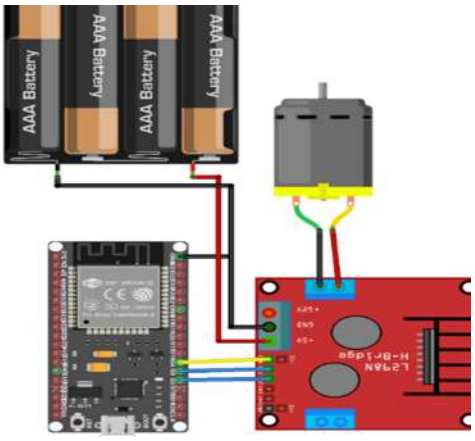
Permet de contrôler la vitesse et la direction des moteurs DC, souvent utilisés dans les robots et les projets de domotique.

- **Commande** : Réagit à la polarité de la tension appliquée
- **Mouvement** : Tourne en continu lorsqu'une tension est appliquée
- **Contrôle** : La vitesse et la direction peuvent être contrôlées en ajustant la tension
- **Alimentation** : Souvent utilisé avec des tensions de 3V, 5V, 12V, etc.
- **Types** : Moteurs à balais (avec balais) et moteurs sans balais (brushless)
- **Couples** : Disponible en différentes gammes de couples en fonction des besoins
- **Contrôle** : Peut être contrôlé en utilisant des ponts en H, des variateurs de vitesse, etc.

IV. Actuateurs (11/18)

E. Moteurs DC (DC Motors)





IV. Actuateurs (12/18)

E. Moteurs DC (DC Motors)

```
// Bibliothèques requises
#include <Arduino.h>

// Broches de connexion au module L298N
const int enA = 2;   // Broche Enable du moteur A
const int in1 = 4;   // Broche d'entrée 1 du moteur A
const int in2 = 5;   // Broche d'entrée 2 du moteur A

const int enB = 15;  // Broche Enable du moteur B
const int in3 = 13;  // Broche d'entrée 1 du moteur B
const int in4 = 12;  // Broche d'entrée 2 du moteur B

// Broches de connexion des boutons
const int boutonGauche = 14; // Bouton pour tourner à gauche
const int boutonDroite = 16; // Bouton pour tourner à droite

void setup() {
  // Configuration des broches
  pinMode(enA, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);

  pinMode(enB, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);

  pinMode(boutonGauche, INPUT_PULLUP);
  pinMode(boutonDroite, INPUT_PULLUP);
}
```

```
void loop() {
  // Lecture de l'état des boutons
  int etatBoutonGauche = digitalRead(boutonGauche);
  int etatBoutonDroite = digitalRead(boutonDroite);

  // Contrôle du moteur en fonction de l'état des boutons
  if (etatBoutonGauche == LOW) {
    // Tourner à gauche
    digitalWrite(enA, HIGH);
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);

    digitalWrite(enB, HIGH);
    digitalWrite(in3, LOW);
    digitalWrite(in4, HIGH);
  } else if (etatBoutonDroite == LOW) {
    // Tourner à droite
    digitalWrite(enA, HIGH);
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);

    digitalWrite(enB, HIGH);
    digitalWrite(in3, HIGH);
    digitalWrite(in4, LOW);
  } else {
    // Arrêter le moteur
    digitalWrite(enA, LOW);
    digitalWrite(enB, LOW);
  }
}
```


IV. Actuateurs (13/18)

F. Moteurs pas à pas (Stepper Motors)

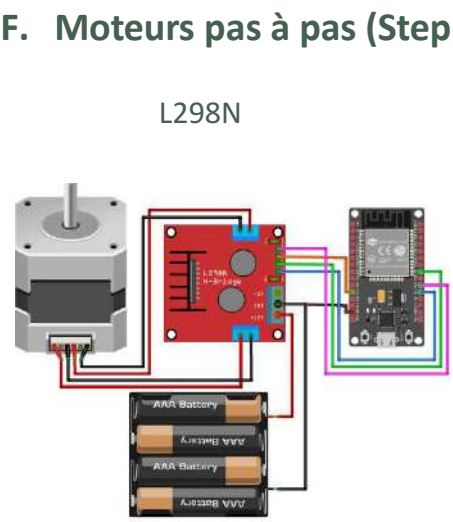
Les moteurs pas à pas sont idéaux pour le contrôle précis du mouvement, ce qui les rend adaptés à une variété d'applications, notamment l'impression 3D, le positionnement précis et les systèmes de suivi solaire.

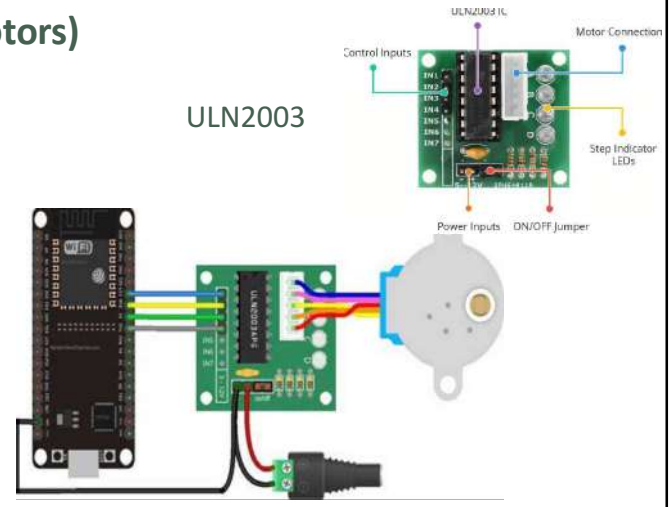
- **Commande** : contrôlé avec des signaux numériques
- **Précision** : Permet un positionnement précis grâce à ses pas individuels
- **Mouvement** : Avance par incréments fixes (pas) plutôt que de tourner en continu
- **Contrôle** : Réagit aux signaux de commande pour chaque pas
- **Alimentation** : Souvent utilisé avec une tension de 5V ou 12V
- **Angle de Pas** : Mesuré en degrés, par exemple 1.8 degrés par pas
- **Couples** : Disponible en différentes gammes de couples en fonction des besoins
- **Contrôle** : Peut être contrôlé en utilisant des pilotes de moteur pas à pas

IV. Actuateurs (14/18)

F. Moteurs pas à pas (Stepper Motors)

L298N ou ULN2003





IV. Actuateurs (15/18)

F. Moteurs pas à pas (Stepper Motors)

L298N ou ULN2003

```
#include <AccelStepper.h>

// Broches de connexion au module L298N
const int enA = 2;   // Broche Enable du moteur A
const int in1 = 4;   // Broche d'entrée 1 du moteur A
const int in2 = 5;   // Broche d'entrée 2 du moteur A

const int enB = 15;  // Broche Enable du moteur B
const int in3 = 13;  // Broche d'entrée 1 du moteur B
const int in4 = 12;  // Broche d'entrée 2 du moteur B

// Broches de connexion des boutons
const int boutonGauche = 14; // Bouton pour tourner à gauche
const int boutonDroite = 16; // Bouton pour tourner à droite

// Création de l'objet AccelStepper
AccelStepper stepper(1, in1, in2); // 1 est le numéro du pilote

void setup() {
  // Configuration des broches
  pinMode(enA, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);

  pinMode(enB, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);

  pinMode(boutonGauche, INPUT_PULLUP);
  pinMode(boutonDroite, INPUT_PULLUP);

  // Configuration du moteur pas à pas
  stepper.setMaxSpeed(1000); // Vitesse maximale
  stepper.setAcceleration(500); // Accélération
}
```

```
void loop() {
  // Lecture de l'état des boutons
  int etatBoutonGauche = digitalRead(boutonGauche);
  int etatBoutonDroite = digitalRead(boutonDroite);

  // Contrôle du moteur pas à pas en fonction de l'état des boutons
  if (etatBoutonGauche == LOW) {
    // Tourner à gauche
    digitalWrite(enA, HIGH);
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);

    digitalWrite(enB, HIGH);
    digitalWrite(in3, LOW);
    digitalWrite(in4, HIGH);
  } else if (etatBoutonDroite == LOW) {
    // Tourner à droite
    digitalWrite(enA, HIGH);
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);

    digitalWrite(enB, HIGH);
    digitalWrite(in3, HIGH);
    digitalWrite(in4, LOW);
  } else {
    // Arrêter le moteur
    digitalWrite(enA, LOW);
    digitalWrite(enB, LOW);
  }
}
```

```
#include <Arduino.h>
#include <Stepper.h>

// Configuration des broches pour le module ULN2003
const int in1 = 14;
const int in2 = 27;
const int in3 = 26;
const int in4 = 25;

// Configuration des boutons
const int boutonGauche = 16; // Bouton pour tourner à gauche
const int boutonDroite = 17; // Bouton pour tourner à droite

// Configuration du moteur pas à pas
Stepper monMoteur(512, in1, in2, in3, in4);

void setup() {
  // Configuration des broches de bouton en entrée avec une résistance de pull-up
  pinMode(boutonGauche, INPUT_PULLUP);
  pinMode(boutonDroite, INPUT_PULLUP);
}

void loop() {
  // Lecture de l'état des boutons
  int etatBoutonGauche = digitalRead(boutonGauche);
  int etatBoutonDroite = digitalRead(boutonDroite);

  // Contrôle du moteur en fonction de l'état des boutons
  if (etatBoutonGauche == LOW) {
    monMoteur.step(512); // Tourner à gauche d'un tour complet (512 pas)
    delay(500); // Délai pour éviter les mouvements rapides
  } else if (etatBoutonDroite == LOW) {
    monMoteur.step(-512); // Tourner à droite d'un tour complet (512 pas)
    delay(500);
  }
}
```

IV. Actuateurs (16/18)

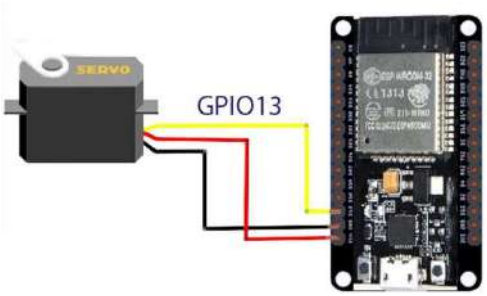
F. Servomoteurs (Servo Motors)

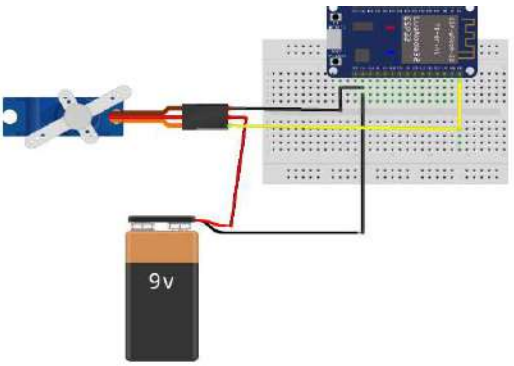
le contrôle précis de l'angle dans les projets de robotique et de mécanique.

- **Commande** : Précise et proportionnelle, généralement via un signal PWM
- **Mouvement** : Rotation précise et limitée dans une plage définie (habituellement 0-180 degrés)
- **Contrôle** : Peut être positionné avec une grande précision et maintenu dans une position donnée
- **Alimentation** : Souvent utilisé avec des tensions de 5V ou 6V
- **Fiabilité** : Fiable pour des mouvements précis et contrôlés
- **Types** : Servomoteurs analogiques et numériques
- **Couples** : Disponible en différentes gammes de couples en fonction des besoins
- **Réponse** : Réagit rapidement aux signaux de commande
- **Angle de Rotation** : La plage de rotation est généralement de 0 à 180 degrés

IV. Actuateurs (17/18)

F. Servomoteurs (Servo Motors)





IV. Actuateurs (18/18)

F. Servomoteurs (Servo Motors)

```
#include <Servo.h>

Servo monServo;  // Créez un objet Servo

const int boutonGauche = 14; // Bouton pour tourner à gauche
const int boutonDroite = 16; // Bouton pour tourner à droite

int position = 90;  // Position initiale du servo

void setup() {
  monServo.attach(2);  // Attachez le servo à la broche 2
  pinMode(boutonGauche, INPUT_PULLUP);
  pinMode(boutonDroite, INPUT_PULLUP);
}
```

```
void loop() {
  int etatBoutonGauche = digitalRead(boutonGauche);
  int etatBoutonDroite = digitalRead(boutonDroite);

  if (etatBoutonGauche == LOW) {
    // Appuyez sur le bouton de gauche pour faire tourner le servo à gauche
    position -= 1;
    if (position < 0) {
      position = 0;
    }
  } else if (etatBoutonDroite == LOW) {
    // Appuyez sur le bouton de droite pour faire tourner le servo à droite
    position += 1;
    if (position > 180) {
      position = 180;
    }
  }

  monServo.write(position);  // Déplacez le servo à la nouvelle position
  delay(15);  // Petite pause pour permettre au servo de se déplacer en douceur
}
```


IV. *Raspberry pi 3P* (1/19)

1. Qu'est-ce qu'une carte Raspberry Pi ?
2. Qu'est-ce qu'on peut faire avec ?
3. Et par rapport à un PC ?
4. Quel OS choisir ?
5. Téléchargement du système : Raspberry Pi OS
6. Créer une carte SD depuis Windows avec Raspberry Pi Imager
7. Premier démarrage
8. Connectez vous en SSH à votre Raspberry Pi pour la contrôller depuis votre ordinateur
9. Comment activer SSH sur le Raspberry Pi
10. Utilisez SSH avec Windows et Putty
11. Raspbian / Quelques commandes