Les Servomoteurs et la Bibliothèque Servo pour ESP32

Cours complet avec exemples pratiques sur Wokwi

4 mai 2025

Table des matières

1 Introduction aux Servomoteurs **3**
- 1.1 Qu'est-ce qu'un servomoteur ? . . . 3
- 1.2 Types de servomoteurs . . . 3
  - 1.2.1 Servomoteurs standard . . . 3
  - 1.2.2 Servomoteurs à rotation continue . . . 3
  - 1.2.3 Servomoteurs numériques vs analogiques . . . 3
- 1.3 Caractéristiques principales . . . 3

2 Principes de fonctionnement **3**
- 2.1 Le signal PWM . . . 3
- 2.2 Circuit interne d'un servomoteur . . . 4

3 L'ESP32 et les servomoteurs **4**
- 3.1 Présentation de l'ESP32 . . . 4
- 3.2 Branchement d'un servomoteur à l'ESP32 . . . 4

4 La bibliothèque ESP32Servo **5**
- 4.1 Présentation de la bibliothèque . . . 5
- 4.2 Installation . . . 5
- 4.3 Fonctions principales . . . 5
- 4.4 Limites et particularités sur ESP32 . . . 5

5 Exemples de code **5**
- 5.1 Exemple 1 : Contrôle basique d'un servomoteur . . . 5
- 5.2 Exemple 2 : Contrôle de plusieurs servomoteurs . . . 6
- 5.3 Exemple 3 : Contrôle précis avec writeMicroseconds() . . . 7
- 5.4 Exemple 4 : Calibration d'un servo . . . 7

6 Implémentation sur Wokwi **8**
- 6.1 Présentation de Wokwi . . . 8
- 6.2 Création d'un projet Wokwi avec servo et ESP32 . . . 8
  - 6.2.1 Étapes pour créer un projet . . . 8
  - 6.2.2 Configuration du projet . . . 8
- 6.3 Exemple complet sur Wokwi . . . 9

7 Applications pratiques **10**
- 7.1 Contrôle d'un bras robotique . . . 10
- 7.2 Système de suivi solaire . . . 12
- 7.3 Station météo avec affichage d'indicateurs . . . 13

8 Problèmes courants et dépannage **15**
8.1 Vibrations et instabilité 15
8.2 Précision et calibration 15
8.3 Consommation électrique 15

9 Optimisations avancées **16**
9.1 Utilisation de plusieurs canaux PWM 16
9.2 Mouvements fluides avec interpolation 17
9.3 Économie d'énergie 18

10 Projets avancés et perspectives **18**
10.1 Contrôle par Wi-Fi ou Bluetooth 18
10.2 Intégration avec des capteurs 20

11 Conclusion **21**

12 Ressources supplémentaires **21**

13 Exercices proposés **21**

1 Introduction aux Servomoteurs

1.1 Qu'est-ce qu'un servomoteur ?

Un servomoteur (ou servo) est un dispositif d'actionnement précis qui permet un contrôle angulaire exact. Il est composé de plusieurs éléments :

- Un moteur à courant continu
- Un système d'engrenages de réduction
- Un circuit de contrôle avec rétroaction
- Un potentiomètre pour mesurer la position angulaire

Contrairement aux moteurs standard, les servomoteurs ne tournent pas continuellement mais se positionnent à un angle précis selon le signal reçu.

1.2 Types de servomoteurs

1.2.1 Servomoteurs standard

Rotation limitée, généralement entre 0° et 180°. Ils sont les plus courants dans les projets électroniques.

1.2.2 Servomoteurs à rotation continue

Modifiés pour tourner continuellement, ils fonctionnent plus comme des moteurs DC avec contrôle de vitesse.

1.2.3 Servomoteurs numériques vs analogiques

- **Analogiques :** Utilisent des circuits analogiques pour le contrôle, moins précis mais moins coûteux.
- **Numériques :** Utilisent des microcontrôleurs pour le traitement du signal, offrant une meilleure précision, une réactivité supérieure et une puissance accrue.

1.3 Caractéristiques principales

- **Couple :** Force de rotation (généralement mesuré en kg-cm ou oz-in)
- **Vitesse :** Temps nécessaire pour tourner de 60° (exprimé en secondes)
- **Tension d'alimentation :** Typiquement 4.8V à 6V
- **Poids et dimensions :** Variables selon les modèles (micro, mini, standard, géant)
- **Angle de rotation :** Généralement 180° pour les servos standard

2 Principes de fonctionnement

2.1 Le signal PWM

Les servomoteurs sont contrôlés par un signal PWM (Pulse Width Modulation ou Modulation de Largeur d'Impulsion). Ce signal se caractérise par :

- Une période généralement de 20ms (50Hz)
- Une largeur d'impulsion variant de 1ms à 2ms
  - 1ms : position 0° (extrême gauche)
  - 1.5ms : position 90° (centre)
  - 2ms : position 180° (extrême droite)

[->] (0,0) – (7,0) node[below] Temps ; [->] (0,0) – (0,3) node[left] Signal ;
[thick] (0,0) – (0,2) ; [thick] (0,2) – (1,2) ; [thick] (1,2) – (1,0) ; [thick] (1,0) – (5,0) ;
[thick] (5,0) – (5,2) ; [thick] (5,2) – (6.5,2) ; [thick] (6.5,2) – (6.5,0) ;
at (0.5,2.3) 1ms ; at (5.75,2.3) 1.5ms ; at (0.5,-0.3) 0° ; at (5.75,-0.3) 90° ;
[<->] (0,-0.8) – (5,-0.8) ; at (2.5,-1.1) 20ms (période) ;

FIGURE 1 – Principe du signal PWM pour contrôle de servomoteur

2.2 Circuit interne d'un servomoteur

Le circuit interne d'un servomoteur fonctionne selon le principe de rétroaction :

1. Le signal PWM est interprété par le circuit de contrôle
2. Ce circuit compare la position désirée (signal PWM) avec la position actuelle mesurée par le potentiomètre
3. Si une différence existe, le moteur est alimenté pour corriger la position
4. Une fois la position atteinte, le moteur s'arrête

3 L'ESP32 et les servomoteurs

3.1 Présentation de l'ESP32

L'ESP32 est un microcontrôleur puissant développé par Espressif Systems, disposant de :
— Processeur dual-core Tensilica Xtensa LX6 cadencé jusqu'à 240 MHz
— Wi-Fi et Bluetooth intégrés
— Nombreuses entrées/sorties (GPIO)
— 16 canaux PWM avec résolution configurable
— Convertisseurs analogique-numérique (ADC)
— Etc.
Cette richesse de fonctionnalités en fait une excellente plateforme pour contrôler des servomoteurs.

3.2 Branchement d'un servomoteur à l'ESP32

Les servomoteurs standard possèdent trois fils :
— Rouge : Alimentation (4.8V - 6V)
— Noir ou Marron : Masse (GND)
— Orange, Jaune ou Blanc : Signal de contrôle PWM

(0,0) rectangle (4,3) ; at (2,1.5) ESP32 ;
(8,0) rectangle (10,2) ; at (9,1) Servo ;
[red, thick] (4,2.5) – (8,1.7) ; [black, thick] (4,1.5) – (8,1) ; [orange, thick] (4,0.5) – (8,0.3) ;
[right] at (4,2.5) 5V ; [right] at (4,1.5) GND ; [right] at (4,0.5) GPIO (PWM) ;
[left] at (8,1.7) VCC (rouge) ; [left] at (8,1) GND (noir) ; [left] at (8,0.3) Signal (orange) ;

FIGURE 2 – Schéma de connexion d'un servomoteur à l'ESP32

Remarques importantes :
— Pour les projets avec plusieurs servos ou des servos puissants, utilisez une alimentation externe
— Connectez toujours les masses (GND) ensemble
— Utilisez n'importe quelle broche GPIO capable de générer un signal PWM sur l'ESP32

4 La bibliothèque ESP32Servo

4.1 Présentation de la bibliothèque

La bibliothèque ESP32Servo est une adaptation de la bibliothèque Servo standard d'Arduino pour l'ESP32. Elle offre une interface simple pour contrôler les servomoteurs tout en tenant compte des spécificités de l'ESP32.

4.2 Installation

Dans l'IDE Arduino :

1. Ouvrez le gestionnaire de bibliothèques (Menu → Croquis → Inclure une bibliothèque → Gérer les bibliothèques)
2. Recherchez "ESP32Servo"
3. Cliquez sur "Installer"

Pour PlatformIO :

```
lib_deps = madhephaestus/ESP32Servo @ ^0.13.0
```

4.3 Fonctions principales

Fonction	Description
attach(pin)	Associe le servo à une broche GPIO spécifique
attach(pin, min, max)	Associe le servo avec des valeurs min/max personnalisées
write(angle)	Positionne le servo à l'angle spécifié (0-180°)
writeMicroseconds(us)	Envoie directement une durée d'impulsion en microsecondes
read()	Lit la position actuelle du servo (en degrés)
attached()	Vérifie si le servo est attaché à une broche
detach()	Détache le servo de la broche

TABLE 1 – Fonctions principales de la bibliothèque ESP32Servo

4.4 Limites et particularités sur ESP32

— L'ESP32 utilise des canaux "ledc" pour générer les signaux PWM
— Par défaut, 16 canaux sont disponibles, donc 16 servos maximum
— La bibliothèque ESP32Servo gère automatiquement l'allocation des canaux
— La fréquence PWM peut être ajustée selon les besoins

5 Exemples de code

5.1 Exemple 1 : Contrôle basique d'un servomoteur

```
#include <ESP32Servo.h>

// Cr ation de l'objet servo
Servo monServo;

// D finition de la broche de signal
int servoPin = 13;

void setup() {
  // Attache le servo    la broche sp cifi e
  monServo.attach(servoPin);
```

```
12
13    // Positionne le servo    la position centrale (90  )
14    monServo.write(90);
15
16    // Attente pour permettre au servo d'atteindre la position
17    delay(1000);
18  }
19
20  void loop() {
21    // Balayage du servo de 0      180
22    for (int angle = 0; angle <= 180; angle += 5) {
23      monServo.write(angle);
24      delay(50);
25    }
26
27    // Balayage du servo de 180      0
28    for (int angle = 180; angle >= 0; angle -= 5) {
29      monServo.write(angle);
30      delay(50);
31    }
32  }
```

5.2 Exemple 2 : Contrôle de plusieurs servomoteurs

```
1   #include <ESP32Servo.h>
2
3   // Cr ation des objets servo
4   Servo servo1;
5   Servo servo2;
6   Servo servo3;
7
8   // D finition des broches
9   int servo1Pin = 13;
10  int servo2Pin = 12;
11  int servo3Pin = 14;
12
13  void setup() {
14    // Attachement des servos
15    servo1.attach(servo1Pin);
16    servo2.attach(servo2Pin);
17    servo3.attach(servo3Pin);
18
19    // Positionnement initial
20    servo1.write(90);
21    servo2.write(90);
22    servo3.write(90);
23
24    delay(1000);
25  }
26
27  void loop() {
28    // S quence de mouvements coordonn s
29    // Servo 1 va de 90      180
30    // Servo 2 va de 90      0
31    // Servo 3 reste    90
32    for (int i = 0; i <= 90; i++) {
33      servo1.write(90 + i);
34      servo2.write(90 - i);
35      delay(15);
36    }
37
38    delay(500);
```

```
39
40    // Retour    la position centrale
41    for (int i = 0; i <= 90; i++) {
42      servo1.write(180 - i);
43      servo2.write(0 + i);
44      delay(15);
45    }
46
47    delay(500);
48  }
```

5.3 Exemple 3 : Contrôle précis avec writeMicroseconds()

```
1  #include <ESP32Servo.h>
2
3  Servo monServo;
4  int servoPin = 13;
5
6  void setup() {
7    Serial.begin(115200);
8    monServo.attach(servoPin);
9
10   // Position centrale
11   monServo.writeMicroseconds(1500);
12   delay(1000);
13 }
14
15 void loop() {
16   // Contr le fin du servo par pas de 50 s
17   // De 1000 s  (0  )    2000 s  (180  )
18   for (int us = 1000; us <= 2000; us += 50) {
19     Serial.printf("Position: %d  s \n", us);
20     monServo.writeMicroseconds(us);
21     delay(500);
22   }
23
24   // Retour    la position initiale
25   monServo.writeMicroseconds(1500);
26   delay(1000);
27 }
```

5.4 Exemple 4 : Calibration d'un servo

```
1  #include <ESP32Servo.h>
2
3  Servo monServo;
4  int servoPin = 13;
5
6  // Valeurs de calibration en microsecondes
7  int minUs = 500;   // Ajustez cette valeur
8  int maxUs = 2400;  // Ajustez cette valeur
9
10 void setup() {
11   Serial.begin(115200);
12
13   // Attach avec param tres de calibration
14   // Param tres: pin, minPulse, maxPulse, angle min (0), angle max (180)
15   monServo.attach(servoPin, minUs, maxUs, 0, 180);
16
17   Serial.println("Servo attach  avec calibration");
18   Serial.printf("Plage: %d  s     %d  s \n", minUs, maxUs);
```

```
19
20    // Test des positions extr mes et milieu
21    Serial.println("Test position 0  ");
22    monServo.write(0);
23    delay(2000);
24
25    Serial.println("Test position 90  ");
26    monServo.write(90);
27    delay(2000);
28
29    Serial.println("Test position 180  ");
30    monServo.write(180);
31    delay(2000);
32  }
33
34  void loop() {
35    // Lecture de commandes depuis le moniteur s rie
36    if (Serial.available() > 0) {
37      int angle = Serial.parseInt();
38      if (angle >= 0 && angle <= 180) {
39        Serial.printf("D placement    %d  \n", angle);
40        monServo.write(angle);
41      }
42    }
43    delay(20);
44  }
```

6 Implémentation sur Wokwi

6.1 Présentation de Wokwi

Wokwi est un simulateur en ligne qui permet de tester des projets électroniques avec différentes plateformes, dont l'ESP32, sans avoir besoin de matériel physique.

6.2 Création d'un projet Wokwi avec servo et ESP32

6.2.1 Étapes pour créer un projet

1. Rendez-vous sur `https://wokwi.com/`
2. Cliquez sur "New Project"
3. Sélectionnez "ESP32"
4. Une fois le projet créé, cliquez sur le bouton "+" pour ajouter des composants
5. Recherchez et ajoutez un "Micro Servo"
6. Connectez les broches du servomoteur à l'ESP32 :
   — Rouge (VCC) → 5V ou 3.3V de l'ESP32
   — Marron/Noir (GND) → GND de l'ESP32
   — Orange/Jaune (Signal) → GPIO de votre choix (ex : GPIO13)

6.2.2 Configuration du projet

Créez un fichier `diagram.json` pour définir les connexions :

```
1  {
2    "version": 1,
3    "author": "Votre Nom",
4    "editor": "wokwi",
5    "parts": [
```

```
    { "type": "wokwi-esp32-devkit-v1", "id": "esp", "top": 0, "left": 0, "attrs": {} },
    { "type": "wokwi-servo", "id": "servo1", "top": 150, "left": 100, "attrs": {} }
  ],
  "connections": [
    [ "esp:GND.1", "servo1:GND", "black", [ "v0" ] ],
    [ "esp:5V", "servo1:V+", "red", [ "v0" ] ],
    [ "esp:D13", "servo1:SIG", "orange", [ "v0" ] ]
  ],
  "dependencies": {}
}
```

6.3 Exemple complet sur Wokwi

Voici le code pour un exemple complet fonctionnel sur Wokwi :

```
#include <ESP32Servo.h>

Servo monServo;
int servoPin = 13;

// Variables pour le contr le interactif
const int potPin = 34;  // Potentiom tre sur GPIO34 (ADC)
int valPot = 0;
int angle = 0;

void setup() {
  Serial.begin(115200);
  Serial.println("Test de servomoteur sur ESP32 avec Wokwi");

  // Configuration du servo
  monServo.attach(servoPin);
  monServo.write(90);  // Position initiale

  Serial.println("Servo initialis     90  ");
  delay(1000);
}

void loop() {
  // Lecture du potentiom tre (si connect  dans Wokwi)
  valPot = analogRead(potPin);

  // Conversion de la valeur (0-4095) en angle (0-180)
  angle = map(valPot, 0, 4095, 0, 180);

  // Affichage des valeurs
  Serial.printf("Potentiom tre: %d, Angle: %d  \n", valPot, angle);

  // Positionnement du servo
  monServo.write(angle);

  // D lai pour  viter  de surcharger le servo
  delay(50);
}
```

Pour ajouter un potentiomètre au projet :

```
// Ajout dans la section "parts" du diagram.json
{ "type": "wokwi-slide-potentiometer", "id": "pot1", "top": 100, "left": 250, "
    attrs": {} }

// Ajout dans la section "connections" du diagram.json
[ "esp:D34", "pot1:SIG", "green", [ "v0" ] ],
```

```
[ "esp:GND.1", "pot1:GND", "black", [ "v0" ] ],
[ "esp:3V3", "pot1:VCC", "red", [ "v0" ] ]
```

7 Applications pratiques

7.1 Contrôle d'un bras robotique

```
#include <ESP32Servo.h>

// Cr ation des objets pour chaque joint du bras
Servo baseServo;    // Rotation de la base
Servo shoulderServo; //  paule
Servo elbowServo;   // Coude
Servo gripperServo; // Pince

// Broches GPIO
const int basePin = 13;
const int shoulderPin = 12;
const int elbowPin = 14;
const int gripperPin = 27;

// Positions pr d finies
struct Position {
  int base;
  int shoulder;
  int elbow;
  int gripper;
};

// Quelques positions pr d finies
Position homePos = {90, 90, 90, 90};
Position pickPos = {45, 45, 30, 90};
Position placePos = {135, 45, 30, 90};

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

  // Attachement des servos
  baseServo.attach(basePin);
  shoulderServo.attach(shoulderPin);
  elbowServo.attach(elbowPin);
  gripperServo.attach(gripperPin);

  // Position initiale
  moveToPosition(homePos);
  delay(2000);
}

void loop() {
  // D mo de s quence pick & place

  // Aller    la position de saisie
  moveToPosition(pickPos);
  delay(1000);

  // Fermer la pince
  closeGripper();
  delay(1000);

  // Retour    la position initiale
  moveToPosition(homePos);
  delay(1000);
```

```
56
57    // Aller    la position de d pose
58    moveToPosition(placePos);
59    delay(1000);
60
61    // Ouvrir la pince
62    openGripper();
63    delay(1000);
64
65    // Retour    la position initiale
66    moveToPosition(homePos);
67    delay(2000);
68  }
69
70  // Fonction pour d placer le bras    une position d finie
71  void moveToPosition(Position pos) {
72    Serial.printf("D placement vers: Base=%d  ,  paule  =%d  , Coude=%d  , Pince=%
        d  \n",
73                  pos.base, pos.shoulder, pos.elbow, pos.gripper);
74
75    // Mouvement progressif pour  viter  les   -coups
76    // (Impl mentation simplifi e, une interpolation plus douce serait
        pr f rable)
77    int steps = 20;
78
79    // Positions actuelles
80    int currentBase = baseServo.read();
81    int currentShoulder = shoulderServo.read();
82    int currentElbow = elbowServo.read();
83    int currentGripper = gripperServo.read();
84
85    // Calcul des incr ments
86    float baseInc = (pos.base - currentBase) / (float)steps;
87    float shoulderInc = (pos.shoulder - currentShoulder) / (float)steps;
88    float elbowInc = (pos.elbow - currentElbow) / (float)steps;
89    float gripperInc = (pos.gripper - currentGripper) / (float)steps;
90
91    // D placement progressif
92    for (int i = 0; i < steps; i++) {
93      baseServo.write(currentBase + (int)(baseInc * i));
94      shoulderServo.write(currentShoulder + (int)(shoulderInc * i));
95      elbowServo.write(currentElbow + (int)(elbowInc * i));
96      gripperServo.write(currentGripper + (int)(gripperInc * i));
97      delay(30);
98    }
99
100   // Position finale exacte
101   baseServo.write(pos.base);
102   shoulderServo.write(pos.shoulder);
103   elbowServo.write(pos.elbow);
104   gripperServo.write(pos.gripper);
105 }
106
107 void openGripper() {
108   Serial.println("Ouverture de la pince");
109   for (int angle = 45; angle <= 90; angle++) {
110     gripperServo.write(angle);
111     delay(15);
112   }
113 }
114
115 void closeGripper() {
116   Serial.println("Fermeture de la pince");
```

```
  for (int angle = 90; angle >= 45; angle--) {
    gripperServo.write(angle);
    delay(15);
  }
}
```

7.2 Système de suivi solaire

```
#include <ESP32Servo.h>

// Cr ation des objets servo
Servo horizontalServo;  // Contr le l'axe horizontal (azimut)
Servo verticalServo;    // Contr le l'axe vertical (  lvation  )

// Broches GPIO
const int horizontalPin = 13;
const int verticalPin = 12;

// Broches pour les photor sistances
const int topLeftLDR = 36;     // ADC1_CH0
const int topRightLDR = 39;    // ADC1_CH3
const int bottomLeftLDR = 34;  // ADC1_CH6
const int bottomRightLDR = 35; // ADC1_CH7

// Variables pour les positions des servos
int horizontalPos = 90;
int verticalPos = 90;

// Sensibilit  du syst me (plus   lev   = moins sensible)
const int threshold = 100;

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

  // Attachement des servos
  horizontalServo.attach(horizontalPin);
  verticalServo.attach(verticalPin);

  // Positionnement initial
  horizontalServo.write(horizontalPos);
  verticalServo.write(verticalPos);

  Serial.println("Syst me de suivi solaire initialis ");
  delay(2000);
}

void loop() {
  // Lecture des capteurs de lumi re
  int topLeft = analogRead(topLeftLDR);
  int topRight = analogRead(topRightLDR);
  int bottomLeft = analogRead(bottomLeftLDR);
  int bottomRight = analogRead(bottomRightLDR);

  // Calcul des moyennes
  int topAvg = (topLeft + topRight) / 2;
  int bottomAvg = (bottomLeft + bottomRight) / 2;
  int leftAvg = (topLeft + bottomLeft) / 2;
  int rightAvg = (topRight + bottomRight) / 2;

  // Ajustement vertical
  if (abs(topAvg - bottomAvg) > threshold) {
    if (topAvg > bottomAvg) {
```

```
      // Plus de lumi re en haut, incliner vers le haut
      verticalPos = constrain(verticalPos - 1, 30, 150);
    } else {
      // Plus de lumi re en bas, incliner vers le bas
      verticalPos = constrain(verticalPos + 1, 30, 150);
    }
    verticalServo.write(verticalPos);
  }

  // Ajustement horizontal
  if (abs(leftAvg - rightAvg) > threshold) {
    if (leftAvg > rightAvg) {
      // Plus de lumi re    gauche, tourner vers la gauche
      horizontalPos = constrain(horizontalPos - 1, 30, 150);
    } else {
      // Plus de lumi re    droite, tourner vers la droite
      horizontalPos = constrain(horizontalPos + 1, 30, 150);
    }
    horizontalServo.write(horizontalPos);
  }

  // Affichage des valeurs pour d bogage
  Serial.printf("LDRs: TL=%d, TR=%d, BL=%d, BR=%d | ",
                topLeft, topRight, bottomLeft, bottomRight);
  Serial.printf("Positions: H=%d  , V=%d  \n",
                horizontalPos, verticalPos);

  // D lai pour stabiliser le syst me
  delay(50);
}
```

7.3 Station météo avec affichage d'indicateurs

```
#include <ESP32Servo.h>
#include <DHT.h>

// Configurations du capteur DHT
#define DHTPIN 15     // Broche de donn es du DHT
#define DHTTYPE DHT22 // Type de capteur (DHT22 / AM2302)

// Cr ation des objets servo
Servo temperatureServo; // Indicateur de temp rature
Servo humidityServo;    // Indicateur d'humidit
Servo pressureServo;    // Indicateur de pression (simul )

// Broches GPIO
const int tempServoPin = 13;
const int humServoPin = 12;
const int pressServoPin = 14;

// Capteur DHT
DHT dht(DHTPIN, DHTTYPE);

// Plages pour le mappage des valeurs
const float tempMin = 0.0;   //  C
const float tempMax = 40.0;  //  C
const float humMin = 0.0;    // %
const float humMax = 100.0;  // %
const float pressMin = 970.0; // hPa (simul )
const float pressMax = 1030.0; // hPa (simul )

// Valeur simul e de pression
```

```
float pressure = 1013.25; // Valeur standard au niveau de la mer

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

  // Initialisation du capteur DHT
  dht.begin();

  // Attachement des servos
  temperatureServo.attach(tempServoPin);
  humidityServo.attach(humServoPin);
  pressureServo.attach(pressServoPin);

  // Positionnement initial
  temperatureServo.write(0);
  humidityServo.write(0);
  pressureServo.write(0);

  Serial.println("Station m t o initialis e");
  delay(2000);
}

void loop() {
  // Lecture des capteurs
  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature();

  // Simulation de changement de pression
  pressure += random(-50, 50) / 100.0;
  pressure = constrain(pressure, pressMin, pressMax);

  // V rification si les lectures sont valides
  if (isnan(humidity) || isnan(temperature)) {
    Serial.println("Erreur de lecture du capteur DHT!");
    return;
  }

  // Mappage des valeurs aux angles des servos (0-180  )
  int tempAngle = map(temperature * 10, tempMin * 10, tempMax * 10, 0, 180);
  int humAngle = map(humidity * 10, humMin * 10, humMax * 10, 0, 180);
  int pressAngle = map((pressure - pressMin) * 10, 0, (pressMax - pressMin) *
    10, 0, 180);

  // Application des angles aux servos
  temperatureServo.write(tempAngle);
  humidityServo.write(humAngle);
  pressureServo.write(pressAngle);

  // Affichage des valeurs
  Serial.printf("Temp rature: %.1f  C  (angle: %d  )\n", temperature, tempAngle)
    ;
  Serial.printf("Humidit : %.1f%% (angle: %d  )\n", humidity, humAngle);
  Serial.printf("Pression: %.2f hPa (angle: %d  )\n", pressure, pressAngle);

  // Attente avant la prochaine mesure
  delay(2000);
}
```

8 Problèmes courants et dépannage

8.1 Vibrations et instabilité

- **Problème :** Le servo vibre ou oscille autour de la position cible.
- **Causes possibles :**
  - Alimentation insuffisante
  - Interférences électriques
  - Charge mécanique trop importante
  - Signaux PWM instables
- **Solutions :**
  - Utiliser une alimentation dédiée pour les servos
  - Ajouter des condensateurs de découplage (100nF, 100µF)
  - Vérifier la capacité de courant de l'alimentation
  - Implémenter un mouvement progressif vers la position cible

8.2 Précision et calibration

- **Problème :** Le servo ne se positionne pas correctement à l'angle demandé.
- **Causes possibles :**
  - Valeurs min/max de PWM incorrectes
  - Tolérance du matériel
  - Friction mécanique
- **Solutions :**
  - Calibrer avec `attach(pin, min, max)`
  - Utiliser `writeMicroseconds()` pour un contrôle plus précis
  - Créer une table de calibration personnalisée

```
// Exemple de table de calibration
const int angleValues[] = {0, 30, 60, 90, 120, 150, 180};
const int microsValues[] = {544, 870, 1180, 1500, 1820, 2130, 2400};

// Fonction d'interpolation pour obtenir la valeur en microsecondes pour un
    angle donn
int getMicrosFromAngle(int angle) {
  // Recherche des bornes les plus proches
  int lowerIndex = 0;
  int upperIndex = 0;

  for (int i = 0; i < sizeof(angleValues)/sizeof(angleValues[0]) - 1; i++) {
    if (angle >= angleValues[i] && angle <= angleValues[i+1]) {
      lowerIndex = i;
      upperIndex = i + 1;
      break;
    }
  }

  // Interpolation lin aire
  float ratio = (float)(angle - angleValues[lowerIndex]) /
                (float)(angleValues[upperIndex] - angleValues[lowerIndex]);

  return microsValues[lowerIndex] +
         (int)(ratio * (microsValues[upperIndex] - microsValues[lowerIndex]));
}
```

8.3 Consommation électrique

- **Problème :** L'ESP32 redémarre lors du mouvement des servos.

— **Causes possibles :**
  — Surconsommation de courant par les servos
  — Alimentation USB insuffisante
  — Pic de courant au démarrage du mouvement
— **Solutions :**
  — Utiliser une alimentation séparée pour les servos
  — Ajouter des condensateurs de découplage de grande capacité (470µF-1000µF)
  — Limiter le nombre de servos actifs simultanément
  — Utiliser un circuit tampon (buffer) ou un transistor pour isoler l'alimentation

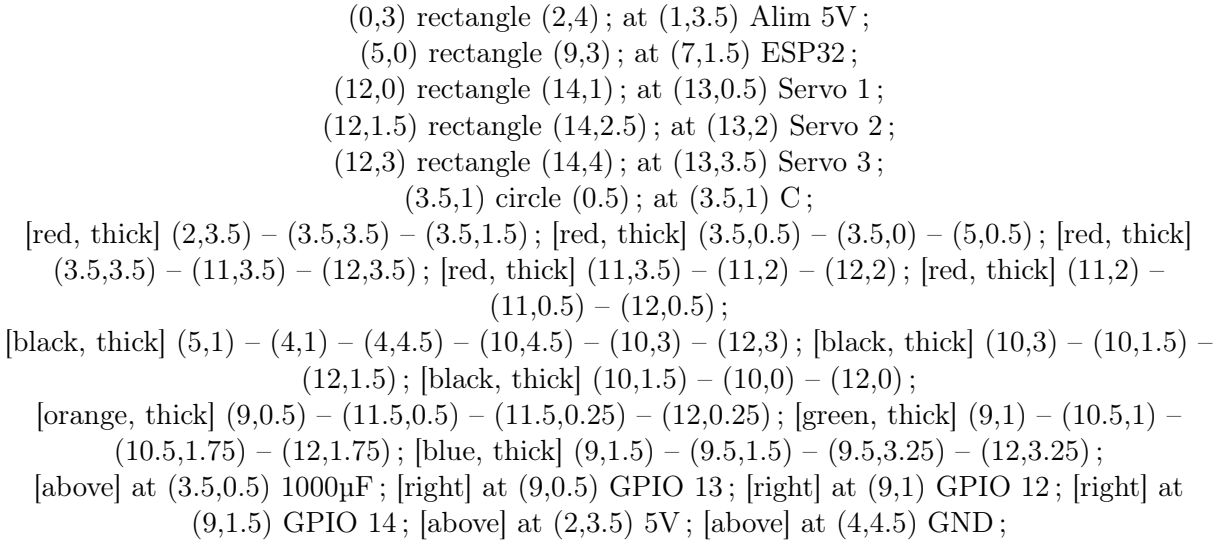

FIGURE 3 – Schéma d'alimentation séparée pour servomoteurs

9 Optimisations avancées

9.1 Utilisation de plusieurs canaux PWM

L'ESP32 dispose de 16 canaux PWM indépendants, permettant un contrôle précis de nombreux servomoteurs :

```
#include <ESP32Servo.h>

// Configuration avanc e des canaux PWM
#define SERVO_CHANNEL_1 1
#define SERVO_CHANNEL_2 2
#define SERVO_RESOLUTION 16   // R solution en bits (plus pr cis)
#define SERVO_FREQ 50         // Fr quence en Hz

// Servos
Servo servo1;
Servo servo2;

// Broches
int servo1Pin = 13;
int servo2Pin = 12;

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

  // Configuration manuelle des canaux
  ESP32PWM::allocateTimer(SERVO_CHANNEL_1);
  ESP32PWM::allocateTimer(SERVO_CHANNEL_2);
```

```
23
24    // Attachement avec param tres personnalis s
25    servo1.setPeriodHertz(SERVO_FREQ);
26    servo2.setPeriodHertz(SERVO_FREQ);
27
28    servo1.attach(servo1Pin, 500, 2400); // Valeurs min/max en  s
29    servo2.attach(servo2Pin, 500, 2400);
30
31    Serial.println("Servos initialis s avec configuration avanc e");
32  }
33
34  void loop() {
35    // Code de contr le des servos
36  }
```

9.2 Mouvements fluides avec interpolation

Pour obtenir des mouvements plus naturels et éviter les à-coups :

```
1   #include <ESP32Servo.h>
2
3   Servo monServo;
4   int servoPin = 13;
5
6   // Position actuelle et cible
7   float currentPos = 90.0;
8   float targetPos = 90.0;
9
10  // Param tres d'interpolation
11  float smoothFactor = 0.05; // Plus petit = plus doux mais plus lent
12
13  void setup() {
14    Serial.begin(115200);
15    monServo.attach(servoPin);
16    monServo.write((int)currentPos);
17  }
18
19  void loop() {
20    // G n ration de cibles al atoires toutes les 3 secondes
21    static unsigned long lastChangeTime = 0;
22    if (millis() - lastChangeTime > 3000) {
23      targetPos = random(30, 150);
24      Serial.printf("Nouvelle cible: %.1f  \n", targetPos);
25      lastChangeTime = millis();
26    }
27
28    // Interpolation de la position
29    if (abs(currentPos - targetPos) > 0.1) {
30      // Calcul de la nouvelle position avec lissage exponentiel
31      currentPos = currentPos + smoothFactor * (targetPos - currentPos);
32
33      // Application au servo
34      monServo.write((int)currentPos);
35
36      Serial.printf("Position: %.1f   -> %.1f  \n", currentPos, targetPos);
37    }
38
39    delay(20); // 50Hz, fr quence adapt e aux servomoteurs
40  }
```

9.3 Économie d'énergie

Pour les projets alimentés par batterie, il est crucial d'optimiser la consommation :

```
#include <ESP32Servo.h>

Servo monServo;
int servoPin = 13;

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

  // Configuration du servomoteur
  monServo.attach(servoPin);

  // Positionnement initial
  monServo.write(90);
  delay(500);

  // D tachement pour   conomiser   l' nergie
  monServo.detach();
  Serial.println("Servo d tach   pour   conomiser   l' nergie ");
}

void loop() {
  // Attente d'une commande s rie
  if (Serial.available() > 0) {
    int angle = Serial.parseInt();

    if (angle >= 0 && angle <= 180) {
      Serial.printf("D placement    %d \n", angle);

      // R attachement du servo
      monServo.attach(servoPin);
      delay(50); // Attente pour stabilisation

      // D placement
      monServo.write(angle);
      delay(500); // Attente pour atteindre la position

      // D tachement pour   conomiser   l' nergie
      monServo.detach();
      Serial.println("Servo d tach ");
    }
  }

  // Mise en veille l g re pour   conomiser   l' nergie
  delay(100);
}
```

10 Projets avancés et perspectives

10.1 Contrôle par Wi-Fi ou Bluetooth

L'ESP32 intègre des capacités Wi-Fi et Bluetooth permettant le contrôle distant des servomoteurs :

```
#include <ESP32Servo.h>
#include <WiFi.h>
#include <ESPAsyncWebServer.h>

// Param tres Wi-Fi
const char* ssid = "VotreSSID";
```

```
const char* password = "VotreMotDePasse";

// Servomoteur
Servo monServo;
int servoPin = 13;
int servoPos = 90;

// Serveur Web
AsyncWebServer server(80);

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

  // Configuration du servo
  monServo.attach(servoPin);
  monServo.write(servoPos);

  // Connexion Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connexion au Wi-Fi...");
  }

  Serial.println("Connect  au Wi-Fi!");
  Serial.print("Adresse IP: ");
  Serial.println(WiFi.localIP());

  // Configuration des routes du serveur web
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    String html = "<html><head><title>Contr le de Servo</title>";
    html += "<meta name='viewport' content='width=device-width, initial-scale
   =1'>";
    html += "<style>body{font-family:Arial;text-align:center;margin:0}";
    html += "input[type=range]{width:80%;margin:20px}</style></head>";
    html += "<body><h1>Contr le de Servomoteur ESP32</h1>";
    html += "<input type='range' min='0' max='180' value='" + String(servoPos) +
    "' oninput='updateServo(this.value)'>";
    html += "<p>Position: <span id='pos'>" + String(servoPos) + "</span>  </p>";
    html += "<script>function updateServo(pos){";
    html += "document.getElementById('pos').innerHTML=pos;";
    html += "var xhr=new XMLHttpRequest();";
    html += "xhr.open('GET','/servo?pos='+pos,true);";
    html += "xhr.send();}</script></body></html>";
    request->send(200, "text/html", html);
  });

  server.on("/servo", HTTP_GET, [](AsyncWebServerRequest *request){
    if (request->hasParam("pos")) {
      servoPos = request->getParam("pos")->value().toInt();
      servoPos = constrain(servoPos, 0, 180);
      monServo.write(servoPos);
      Serial.printf("Nouvelle position servo: %d  \n", servoPos);
    }
    request->send(200, "text/plain", "OK");
  });

  // D marrage du serveur
  server.begin();
}

void loop() {
  // Le traitement des requ tes est g  r   de mani re asynchrone
```

```
}
```

10.2 Intégration avec des capteurs

```
#include <ESP32Servo.h>
#include <Wire.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>

// Objets
Servo servoX;
Servo servoY;
Adafruit_MPU6050 mpu;

// Broches
const int servoXPin = 13;
const int servoYPin = 12;

// Variables
float accX, accY, accZ;
int angleX = 90;
int angleY = 90;

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

  // Initialisation du MPU6050
  if (!mpu.begin()) {
    Serial.println("Erreur d'initialisation du MPU6050!");
    while (1) {
      delay(10);
    }
  }

  // Configuration du MPU6050
  mpu.setAccelerometerRange(MPU6050_RANGE_2_G);

  // Configuration des servos
  servoX.attach(servoXPin);
  servoY.attach(servoYPin);

  // Position initiale
  servoX.write(angleX);
  servoY.write(angleY);

  Serial.println("Syst me d'  quilibrage   initialis ");
}

void loop() {
  // Lecture des donn es du capteur
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);

  accX = a.acceleration.x;
  accY = a.acceleration.y;
  accZ = a.acceleration.z;

  // Calcul des angles d'inclinaison
  // Formule simplifi e pour convertir l'acc l ration en angle
  int newAngleX = map(constrain(accY, -10, 10) * 10, -100, 100, 45, 135);
  int newAngleY = map(constrain(accX, -10, 10) * 10, -100, 100, 45, 135);

```

```
  // Filtrage pour  viter  les mouvements brusques
  angleX = 0.95 * angleX + 0.05 * newAngleX;
  angleY = 0.95 * angleY + 0.05 * newAngleY;

  // Application aux servos
  servoX.write(angleX);
  servoY.write(angleY);

  // Affichage des valeurs
  Serial.printf("Acc: X=%.2f, Y=%.2f, Z=%.2f | Angles: X=%d  , Y=%d  \n",
                accX, accY, accZ, angleX, angleY);

  delay(20);
}
```

11 Conclusion

Les servomoteurs constituent des actionneurs polyvalents parfaitement adaptés aux projets d'électronique embarquée. Associés à l'ESP32 et à la bibliothèque ESP32Servo, ils offrent des possibilités quasi illimitées pour la création de systèmes automatisés, robotiques ou interactifs.

Dans ce cours, nous avons exploré :

— Les principes fondamentaux des servomoteurs et leur fonctionnement
— L'utilisation de la bibliothèque ESP32Servo pour contrôler facilement les servos
— Des exemples pratiques d'implémentation dans Wokwi
— Des applications variées, du contrôle basique à des projets complexes
— Des techniques avancées pour optimiser les performances et la fiabilité

La maîtrise des servomoteurs constitue une compétence clé pour tout projet impliquant du mouvement contrôlé. Avec les connaissances acquises dans ce cours, vous disposez désormais des bases nécessaires pour intégrer ces composants dans vos propres créations.

12 Ressources supplémentaires

— GitHub de la bibliothèque ESP32Servo
— Simulateur Wokwi
— Guide de sélection de servomoteurs
— Tutoriels ESP32 et servo

13 Exercices proposés

1. Créez un système de verrouillage contrôlé par RFID utilisant un servomoteur.
2. Implémentez un système d'alarme avec détection de mouvement qui utilise un servo pour orienter une caméra.
3. Concevez un mini-robot marcheur à six pattes utilisant trois servomoteurs.
4. Développez une interface de contrôle Bluetooth pour positionner précisément un servomoteur.
5. Créez un système de mesure de la qualité de l'air qui affiche les niveaux sur des cadrans actionnés par servos.