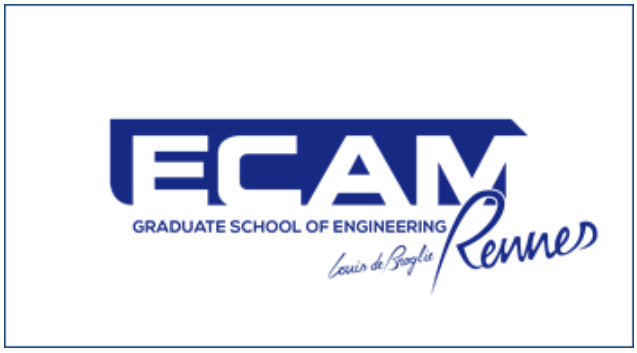


ECAMBOT

- Tips And Tricks -

2020-2021

ECAM 4

◊ ◊ ◊

V2.0

Hervé GUIBOURG <herve.guibourg@ecam-rennes.com>

Denys BOITEAU <denys.boiteau@ecam-rennes.fr>

Historique des versions

Version	Date	Section ajoutées ou modifiés
v1	12/03/2021	Création du document
v2	16/03/2021	§2.2.1: « V2 : Attention … » Ajout d'information sur les limitations sur le choix de la broche RX des cartes arduino pour la communication Bluetooth. §2.6 : ***Le pont en H & PWM*** – Utilisation d'un composant simple pour le pilotage d'un moteur CC. Ce composant est à privilégier pour les actions autre que le roulage, en utilisant des moteurs de base accessible dans la zone magasin (rappel encore une fois : pour les actions basiques, ne pas utiliser les moteurs avec codeur intégrés).

1 Généralités – objet du document

1.1 Introduction

Ce document liste des pratiques et retours d'expériences sur des éléments permettant de vous aider dans l'avancement technique de votre robot.

Ce document s'enrichira au fur et à mesure du temps, merci d'avance pour vos retours/besoin.

Important : Il est à noter qu'il n'y a aucune obligation d'implémenter la totalité des informations qui seront décrit ici et donc **ce document n'est pas un manuel de fabrication d'un robot type**.

1.2 Règle de nommage d'un code

Déjà vu lors des BE d'architecture des systèmes.

Il n'y a aucune obligation pour cela, mais ce sont de bonnes habitudes à acquérir.

1.3 Le C++ et les librairies sur Arduino

Il n'y a aucune obligation, mais la programmation objet utilisée pour un robot est assez simple et permet de faciliter la lecture du code et donc d'optimiser la mise au point logicielle, comme on a pu le voir lors du BE3 de l'architecture des systèmes.
ES8 → https://moodle.ecam-rennes.fr/mod/folder/view.php?id=13786
AS7 → https://moodle.ecam-rennes.fr/mod/folder/view.php?id=13787

Le C++ et le Java sont très proches.
Pour un robot, tous les éléments à piloter sont présent tout le temps aussi, les déclarations des variables, objets sont forcément statique. Et donc ici en C++ il n'y aura pas la nécessité d'implémenter les destructeurs des classes C++. Le constructeur quant à lui est nécessaire et appelé lors de la création des objets statique, c'est à dire AVANT l'appel à la fonction setup() et donc dans un constructeur Arduino, il n'est pas possible d'utiliser des print ou println pour le débug sur le moniteur série.
Toutes les librairies disponibles pour les cartes Arduino sont codées en C++
Il est aussi possible d'utiliser ces classes C++ en local en copiant les fichiers .cpp et .h dans le même dossier que le fichier pricipal .ino (Voir BE3 avec les liens ci-dessus)

- Fichier .h : La structure C++ de la classe avec le prototype des méthodes des objets et leurs niveau d'acces (`public` ou `private`). Les variables locales à la class doivent être '`private`' : Ce sont les bonnes pratiques.
- Fichier .cpp : Le code C++ des méthodes de la classe

Un exemple avec la Class `Digital` pour piloter un I/O du processeur de la carte Arduino :

1.3.1 Digital.h / Class C++

```
class Digital
{
  public:
    Digital(int pin_Digital, int direction); // Constructeur
    void On();           // = Set(High) to set '1' to I/O pin
    void Off();          // = Set(Low)  to set '0' to I/O pin
    void Set(int val);   // to set val to I/O pin
    void Toggle();       // to invert value of OUTPUT I/O pin
    int  Get();          // to get valut from I/O pin
  private:
    int _pin_Digital;
    int _val;             // I/O value
};
```

1.3.2 Utilisation dans le fichier .ino

- Création des objets statiques LED_buildIn, LED_R, LED_B, buzzer

```
#include "Digital.h"          // For LED & Buzzer

static Digital   LED_builtIn (LED_BUILTIN, OUTPUT);      // LED on board
static Digital   buzzer      (pin_Buzzer , OUTPUT);      // Buzzer
```

- Exemple de codage pour faire clignoter la LED «LED_BUILTIN» dans le fichier .ino

```
../..
LED_builtIn.On();  // La LED builtIn est allumée
delay(100);
LED_builtIn.Off(); // La LED builtIn est éteinte
delay(100);
../..
```

2 Tips And Tricks

2.1 Le capteur Ultrason HC-SR04

Le capteur HC-SR04 utilise les ultrasons pour déterminer la distance d'un objet. Il offre une excellente plage de détection sans contact, avec des mesures de haute précision et stables. Son fonctionnement n'est pas influencé par la lumière du soleil ou des matériaux sombres. Il est alimenté en 5V.

La gestion de ce capteur provient d'une librairie qu'il faudra installer.
Librairie en ligne, par exemple : https://github.com/gamegine/HCSR04-ultrasonic-sensor-lib

2.1.1 HCSR04.h / Class C++

Utilisé dans la librairie sous github (voir ci-dessus)

2.1.2 Utilisation dans le fichier .ino

- Création de l'objet statique «hc»

```
#include <HCSR04.h>     // HC-SR04

static HCSR04 hc(pin_hc_trig, pin_hc_echo); // Ultrasonic HCSR04
```

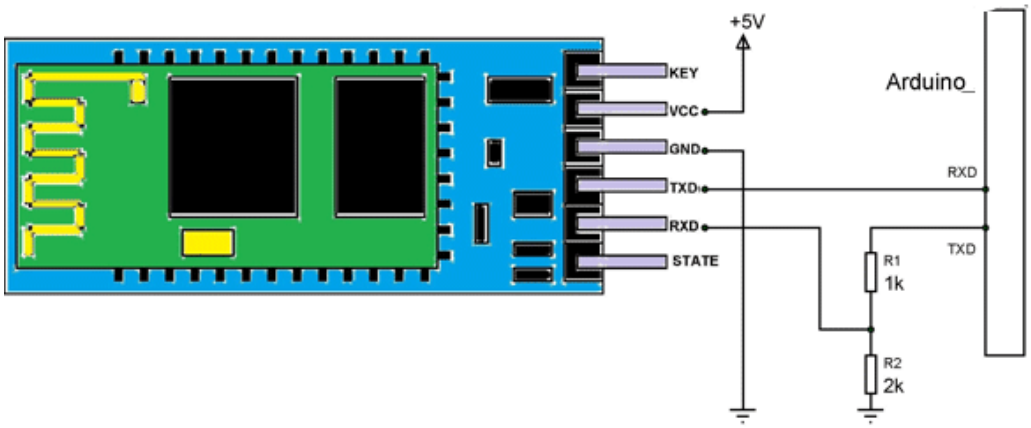
- Exemple de codage dans le fichier .ino

```
if (hc.dist()<5 ) { // ultrasonic sensor: the robot detects object
  motor.turn(10); // Turn speed 10%
  gyro.WaitAngle(90); // until 90°
}else{ // The horizon is free
  ../..  // Let's go !
}
```

2.2 La communication bluetooth HC-05

2.2.1 Généralités et précautions d'emploi

Ce module permet la communication Bluetooth avec un lien série (UART).



Ce module Bluetooth utilise un lien de communication Série (UART) pour récupérer les caractères venant d'un Smartphone (Utilisation de la librairie <SoftwareSerial.h>).

De nombreux tutos existent sur internet, **et beaucoup indiquent des erreurs** :

- **Le module s'alimente en 5V** (entre 3.6V et 6V : c'est indiqué sur le module) **et pas 3.3V**

Les I/O TX et RX du module sont en 3.3V (c'est aussi indiqué sur le module) et pas 5V. Dans le sens Bluetooth TX (3.3V) → Arduino RX (5V max) cela n'est pas génant, mais dans le sens Arduino TX (5V) → Bluetooth RX (3.3V max) cela peut être destructeur.

- le besoin du robot, seul le sens Smartphone → Bluetooth → Arduino est nécessaire et donc Bluetooth pin TX → Arduino pin RX avec une liaison directe.

Le sens Arduino → Bluetooth → Smartphone n'est pas nécessaire. Si ce sens est utilisé, les résistances R1 et R2 doivent être ajoutées pour rabaisser la tension vers le module en dessous des 3.3V. Ce sens Arduino → Bluetooth peut également être utilisé pour piloter le module en commandes AT afin de reconfigurer le module (son nom par exemple). Mais c'est une autre histoire non décrite ici, car une fois de pas forcément nécessaire pour ECAMBOT.

Sur la carte Arduino, les broches 0 (RX) et 1 (TX) sont partagées avec le USB, et il faudra éviter que le module Bluetooth soit connecté à ces broches ou on aura des problèmes pour télécharger le code vers la carte Arduino, d'où l'utilisation de la librairie SoftwareSerial.

```
   #include <SoftwareSerial.h>

   SoftwareSerial myBLESerial(14, 15); // RX de l'arduino, TX de l'arduino

   // TX n'est pas utilisé pour l'utilisation faite avec le robot ECAMBOT, mais
nécessaire pour le constructeur de l'objet  myBLESerial
```

V2 : Attention, il y a des restrictions sur le choix des broches Arduino RX pour certaine version de carte comme Arduino MEGA par exemple.
Voir la section Limitations sous le lien de référence :
https://www.arduino.cc/en/Reference/SoftwareSerial

2.2.2 Utilisation dans le fichier .ino

- Création de l'objet myBLESerial pour la communication Bluetooth
  et l'objet Serial pour la communication USB

```
#include <SoftwareSerial.h>

SoftwareSerial myBLESerial(14, 15); // RX, TX

void setup()
{
  myBLESerial.begin(9600); // Init du lien série pour le module Bluetooth
  Serial.begin(9600);      // Init du lien série pour le moniteur de débug
  while (! Serial);        // Attente de la fin d'initialisation série
  Serial.println("fin du setup");
} // end setup

void loop() {
  if (myBLESerial.available()>0) {
    char rxBluetoothMsg = (char)myBLESerial.read();
    decode(rxBluetoothMsg);
   }
}

void decode(char rxBluetoothMsg) {
  if (rxBluetoothMsg=='a') {Serial.print("a"); /* A compléter */ }
  if (rxBluetoothMsg=='b') {Serial.print("b"); /* A compléter */ }
  ../..
}
```

2.3 *Servomoteur - SG90*

Un servomoteur permet de faire pivoter un axe d'un angle compris entre 0 et 180°.
Un modèle assez utilisé est le SG90
https://www.arduino.cc/en/Tutorial/Knob
Mais il en existe d'autres permettant de manipuler jusqu'à 60 Kg.

Ce servomoteur est tellement populaire dans le monde Arduino, qu'il n'est pas nécessaire d'installer une librairie pour le gérer.

Câblage
- Noir : GND
- Rouge : 5V
- Jaune : PWM signal

2.3.1 Paramétrage

Juste un signal de commande par servomoteur utilisé.

2.3.2 Utilisation dans le fichier .ino

Exemple avec un servomoteur :

- Création des objets statique `servo1`

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

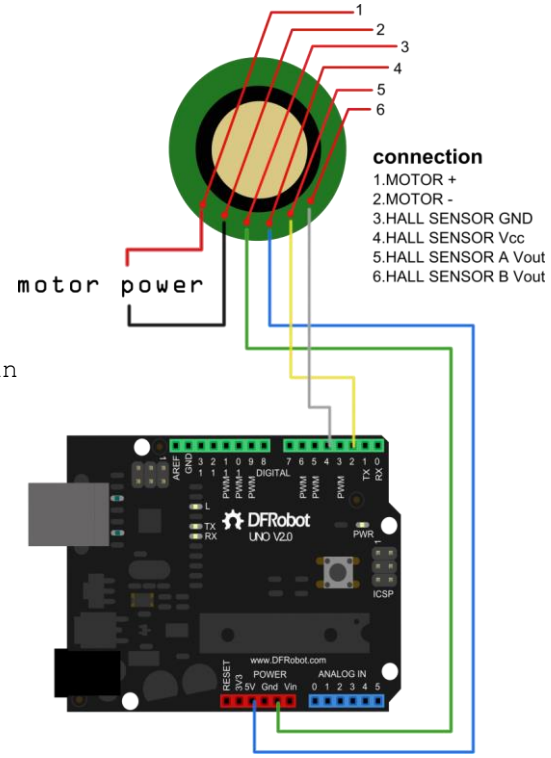
```
static Servo servo1;

int posX= 43; // entre 0 et 180
int posY=152; // entre 0 et 180

void setup() {
  servoX.attach(10); // Affectation de la broche 10 pour servoX
}

void loop() {

  ../..

  servo1.write(posX); // Positionne le servomoteur avec l'angle posX
  delay(1000); // 1 sec
  servo1.write(posY); // Positionne le servomoteur avec l'angle posY
  delay(1000); // A sec

  ../..

}
```

Dans ce code, le servomoteur se déplace toutes le secondes entre la position à 43° et 152°
Si vous souhaitez avoir un déplacement plus lent du servomoteur, vous devez utiliser une boucle 'for' pour faire varier l'angle progressivement et ajouter une petite temporisation dans la boucle.

2.4 Moteur 12V

Le moteur est le type Gear Motor w/Encoder model No.GB37Y3530-12V-251R.
Câblage :
→ 12V et GND sur les fils 1 & 2
→ 5V et GND sur les fils 4 & 3
→ Pin #2 & #3 pour le capteur à effet hall

1.	MOTOR+	+12V
2.	MOTOR-	GND
3.	HALL SENSOR GND	GND
4.	HALL SENSOR Vcc	+5V
5.	HALL SENSOR A Vout	Pin#2
6.	HALL SENSOR B Vout	Pin#4



Nota1 : Pin#2 = Int0 sur UNO
Nota2 : Pin#3 = Int1 sur UNO (réservé pour un second moteur)

```
//The sample code for driving one way motor encoder
const byte encoder0pinA = 2; // interrupt pin 0
const byte encoder0pinB = 4; // digital pin 4
volatile byte        encoder0PinALast;
volatile int         duration;   // number of the pulses
volatile boolean     direction; // rotation direction

void setup() {
  Serial.begin(115200);//Initialize the serial port
  while (! Serial);
  //Initialize the module
  direction = true; //default -> Forward
  pinMode(encoder0pinB,INPUT);
  attachInterrupt(0, wheelSpeed, CHANGE); // Init of Int0 (Pin #2)
}

void loop() {
  Serial.print("Pulse:");
  Serial.println(duration);
```

```
  duration = 0;
  delay(100);
}

void wheelSpeed() // Int0 (Pin #2 changed) {
  int Lstate = digitalRead(encoder0pinA);
  if ((encoder0PinALast == LOW) && Lstate==HIGH) {
    int val = digitalRead(encoder0pinB);
    if (val == LOW && direction) {
      direction = false; //Reverse
    }else if (val == HIGH && !Direction) {
      direction = true; //Forward
    }
  }
  encoder0PinALast = Lstate;
  if (!direction) duration++;
  else duration--;
}
```

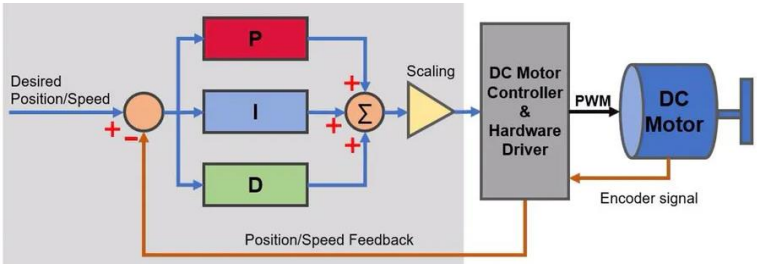
Nota1 : volatile est important pour éviter que le compilateur optimise de trop le code
Nota2 : attachInterrupt perment de gérer une interruption en appelant la fonction wheelSpeed a chaque changement de tension constaté sur la broche Int0 (Pin #2)

2.5 Le PID

http://www.telecom-robotics.org/wiki/Tutoriels/AsservissementPid/WebHome

http://www.ferdinandpiette.com/blog/2011/08/implementer-un-pid-sans-faire-de-calculs/

Le **régulateur PID**, appelé aussi **correcteur PID** (**P**roportionnel, **I**ntégral, **D**érivé) est un système de contrôle permettant d'améliorer les performances d'un asservissement, c'est-à-dire un système en boucle fermée.



Le **PID** améliore 3 principales caractéristiques d'un système : la rapidité, la précision et la stabilité.
C'est le régulateur le plus utilisé dans l'industrie où ses qualités de correction s'appliquent à de multiples grandeurs physiques.
Le correcteur PID agit de trois manières : action **Proportionnelle,** action **Intégrale,** action **Dérivée**.

2.5.1 Exemple avec un chauffage électrique

Une régulation consiste à obliger un système à suivre une consigne. Prenons le chauffage d'une pièce avec un radiateur à puissance variable (on contrôle son courant)
On crée une régulation. L'asservissement en boucle fermée surveille à tout moment la température de la pièce et la compare avec la consigne. Donc en fonction de la différence entre température pièce et consigne (température voulue) on va générer un signal de sortie (intensité du radiateur) qui sera :
Proportionnel à la différence (température consigne - température pièce) pour la rejoindre plus vite et mettre le chauffage à fond si nécessaire.

```
error    = setpoint – measured_value
output = Kp * error
```

Lorsque l'on s'approche de la consigne on risque de dépasser la température cible (inertie du chauffage qui ne s'arrêtera pas de suite…). Donc on surveille l'évolution en imaginant le futur (**Intégration**). Donc même si on dépasse la consigne, l'**Intégrateur** prévoyant le futur oblige le

système à adapter la sortie pour mettre la pièce à température de la consigne. Seulement cette précision grève la rapidité d'obtention.

```
output = Ki * integral
```

Pour éviter cela on surveille la pente (**Dérivée**) pour vérifier si on ne dépasse pas la consigne. En cas de risque on introduit une correction.

```
output = Kd * derivative
```

L'ensemble s'appelle un correcteur **PID**.

```
output = Kp * error + Ki * integral + Kd * derivative
```

Les trois coefficients Kp, Ki, Kd permettent d'ajuster le système.

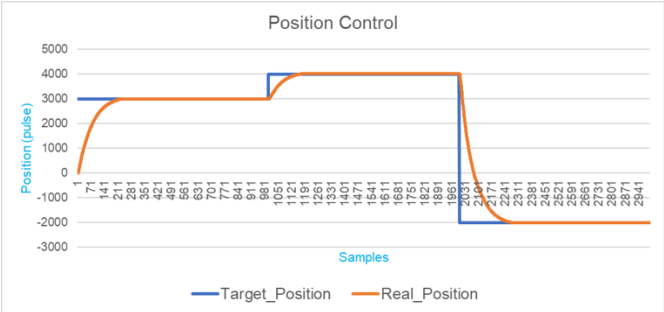
Attention, les coefficients Ki et Kd dépendent de la fréquence d'échantillonnage du système ! En effet, l'intégrateur fera la somme des erreurs au cours du temps ! Si on échantillonne deux fois plus vite, on sommera deux fois plus d'échantillons. Du coup, le coefficient Ki devra être divisé par 2. A l'inverse, pour le dérivateur, si on double la fréquence d'échantillonnage, il faudra doubler le coefficient Kd afin de garder les mêmes performances du PID. Plus la fréquence d'échantillonnage est élevée et plus le PID sera performant. (En effet, plus on échantillonne souvent et plus l'intégration et la dérivée seront précises).

2.5.2 PID.h / Class C++

```
class PID
{
  public:
    PID(float Kp, float Ki, float Kd);
    float computePID(float setpoint, float measured_value);
};
```

2.5.3 Paramétrage

Les coefficients `Kp`, `Ki`, `Kd` sont envoyés au constructeur de l'objet lors de sa création.



Quelques références :

http://www.ferdinandpiette.com/blog/2011/08/implementer-un-pid-sans-faire-de-calculs/
http://www.telecom-robotics.org/wiki/Tutoriels/AsservissementPid/WebHome

2.5.4 Utilisation dans le fichier .ino

- Création de l'objet statique pidL, pidR

```
#include "PID.h"                // PID
// PID Objects...
#define _Kp_ 0.3  // PID: Kp
#define _Ki_ 0.05 // PID: Ki
#define _Kd_ 0.0  // PID: Kd
PID   pidL(_Kp_, _Ki_, _Kd_); // PID: set Kp, Ki, Kd
PID   pidR(_Kp_, _Ki_, _Kd_); // PID: set Kp, Ki, Kd
float pidOutL = 0;
float pidOutR = 0;
```

- Exemple de codage dans le fichier .ino

```
  if       (!bL &&  bF && !bR) { setL = 20; setR = 0;  // move right
  }else{
```

```
    ../..
  }
  pidOutL = pidL.computePID(setL, pidOutL); // setpoint, measured_value
  pidOutR = pidR.computePID(setR, pidOutR); // setpoint, measured_value
  motor.arc(pidOutL,pidOutR);
```

Attention, le code fournit au BE3 n'a à ma connaissance pas été validé, il se peut que des ajustements soient nécessaires.

2.6 Le pont en H & PWM

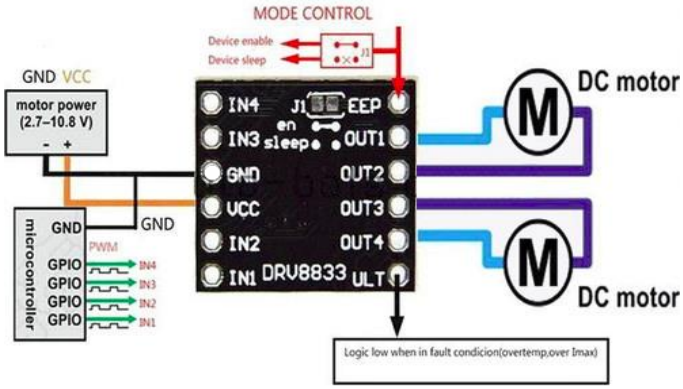
Remarque préliminaire : cette section parle du composant imagé ci-dessous. D'autres composants de même type sont disponibles en magasin (marque pololu), et les codes sources associés sont dispos sur Moodle.

Un pont en H (composant : DRV8833) permet de changer la direction du courant pour deux moteurs indépendants comme vu lors des BE d'architecture. Ici : 1.2A max et 2.7V à 10.8V de tension.
Attention : Si vous utilisez un moteur 12V, ce module DRV8833 ne convient pas
Ce pont en H permet de changer le sens de rotation du moteur.
De plus ce module accepte la variation de vitesse en PWM.
Pour PWM : utiliser analogWrite() voir :
https://www.arduino.cc/en/Tutorial/Foundations/PWM
Attention, toutes les broches Arduino n'acceptent pas la fonction PWM, voir sous :
https://www.locoduino.org/spip.php?article47
Ces pattes PWM sont à brancher sur les entrées IN1, IN2 ou/et IN3/IN4.

Attention : Pour activer ce module, un pont de soudure doit être fait sur J1 (marquage « EEP » en haut du module)

2.6.1 Gestion d'un monteur un pont en H

Nous allons ici piloter un moteur DC avec un module de contrôle DRV8833.



Ce pont en H est capable de délivrer continuellement 1.2A sur chaque canal. Il inclut les protections suivantes:

- contre la polarisation inverse,
- la sous-tension,
- les courants trop importants,
- et les températures trop élevées

Le choix de la tension du module se fait en fonction du besoin du moteur (**Par exemple 6V si c'est la tension du moteur**). Utiliser une batterie externe qui servira d'alimentation de la carte Arduino sur la broche Vin. En cas de besoin de 3.3V ou 5V pour vos périphériques, ces tension sont disponible sur la carte.
L'interface avec le module de contrôle est constituée de deux signaux (AIN1, AIN2 pour le moteur #1 et BIN1, BIN2 pour le moteur #2) – Noté sur le module ci-dessus IN1, IN2, IN3, IN4.

xIN1	xIN2	xOUT1	xOUT2	Fonction
0	0	Z	Z	Roue Libre
0	1	L	H	Arrière
1	0	H	L	Avant
1	1	L	L	Frein

2.6.2 Utilisation dans le fichier .ino

Voir le codage vu lors du BE3 : dans Motor.cpp

ES8 → https://moodle.ecam-rennes.fr/mod/folder/view.php?id=13786
AS7 → https://moodle.ecam-rennes.fr/mod/folder/view.php?id=13787

- FIN DU DOCUMENT -