Informatique Industrielle

Les systèmes embarqués

Laurent Barbé (laurent.barbe@unistra.fr)
http://eavr.u-strasbg.fr/~barbe

INSA Strasbourg, spécialité mécatronique

Sommaire

Contents

Avant-propos

Qui suis-je ?

Laurent Barbé, ingénieur de recherche en contrôle-commande des systèmes robotiques au Laboratoire ICube équipe Automatique, Vision et Robotique (AVR) et rattaché à la plate-forme Imagerie Interventionnelle et Robotique Médicale (I2RM).

Qu'est ce que je fais?

- Contrôle-commande pour des systèmes robotiques dans le domaine médical.
- Programmation systèmes embarqués (contraintes temps-réel), intégration et commande de solutions d'actionnement pour la robotique

Avant-propos

Comment allons-nous travailler ?

1. Introduction et rappel terminologie et notions d'informatique
2. Les notions de bases sur les contraintes temps-réel pour les systèmes embarqués
3. Présentation du système que vous allez utiliser pour le projet
4. Projet ...
5. Exposé des résultats obtenus

Avant-propos

Organisation du cours

- 4 séances de cours/TD
- 4 séances de suivi de projet par groupe (intervenants L. Barbé et C. Roth):
  - Groupe 1 : étudiants Master
  - Groupe 2 : étudiants non Master
- dernière séance évaluation des résultats obtenus au cours du projet

Le projet

- par groupe de 3 à 4 étudiants ;
- responsabilité des maquettes ;
- évaluation (par groupe) : Présentation du matériel, méthodes, résultats et analyse.

Avant-propos

Pré-requis

- Modélisation de systèmes robotique (MGD, MGI) ;
- Simulation avec les outils Matlab ;
- Notions de base en programmation informatique (C/C++) ;
- Architecture des ordinateurs ;
- Mise en oeuvre de systèmes numériques (échantillonnage) et automatique.

Contents

L'informatique industrielle

Définition

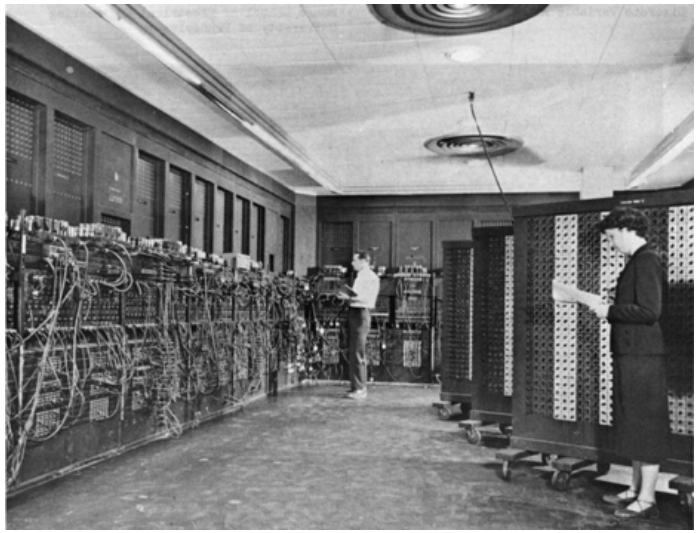
- l'informatique industrielle développe des interfaces entre l'informatique et les appareils industriels.
- les outils informatiques, qui vont de l'analyse à la programmation, sont alors mis en œuvre pour pour réaliser l'interface entre l'informatique, l'électronique, la mécanique, la robotique, l'électrotechnique, etc. à vocation industrielle et qui ne sont pas uniquement à base d'ordinateurs.

(inspirée de la définition Wikipédia)

Evolution des systèmes informatiques

1ère génération : Calculateurs de la taille d'un bâtiment

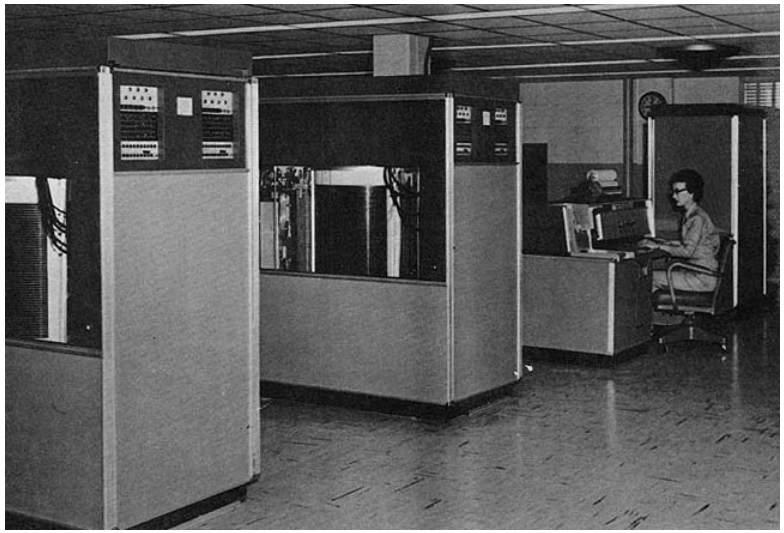
Mark 1 Harvard en 1944



ENIAC en 1946

Evolution des systèmes informatiques

2ième génération : Avènement des transistors

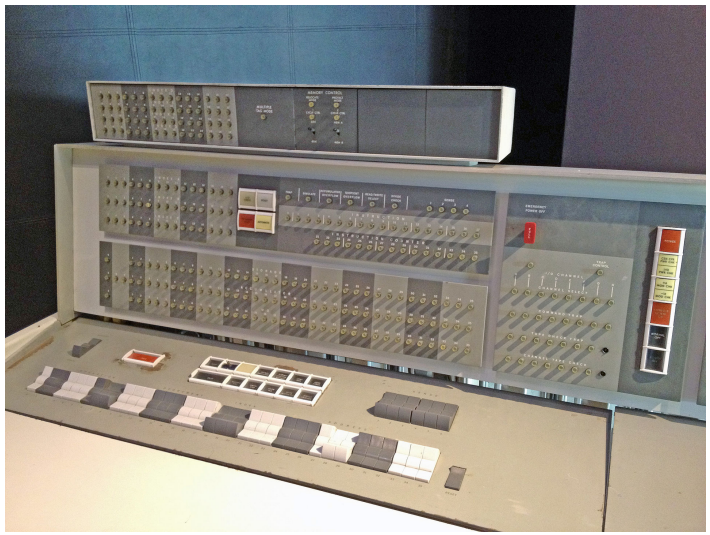


RAMAC 350 en 1956

IBM 1620 en 1959

Evolution des systèmes informatiques

3ième génération : Calculateurs à taille humaine



IBM 7094 en 1962

IBM 360/20 en 1965

Evolution des systèmes informatiques

4ième génération : Les premiers ordinateurs personnels de bureau

Apple II en 1977

IBM PC 5150 en 1981

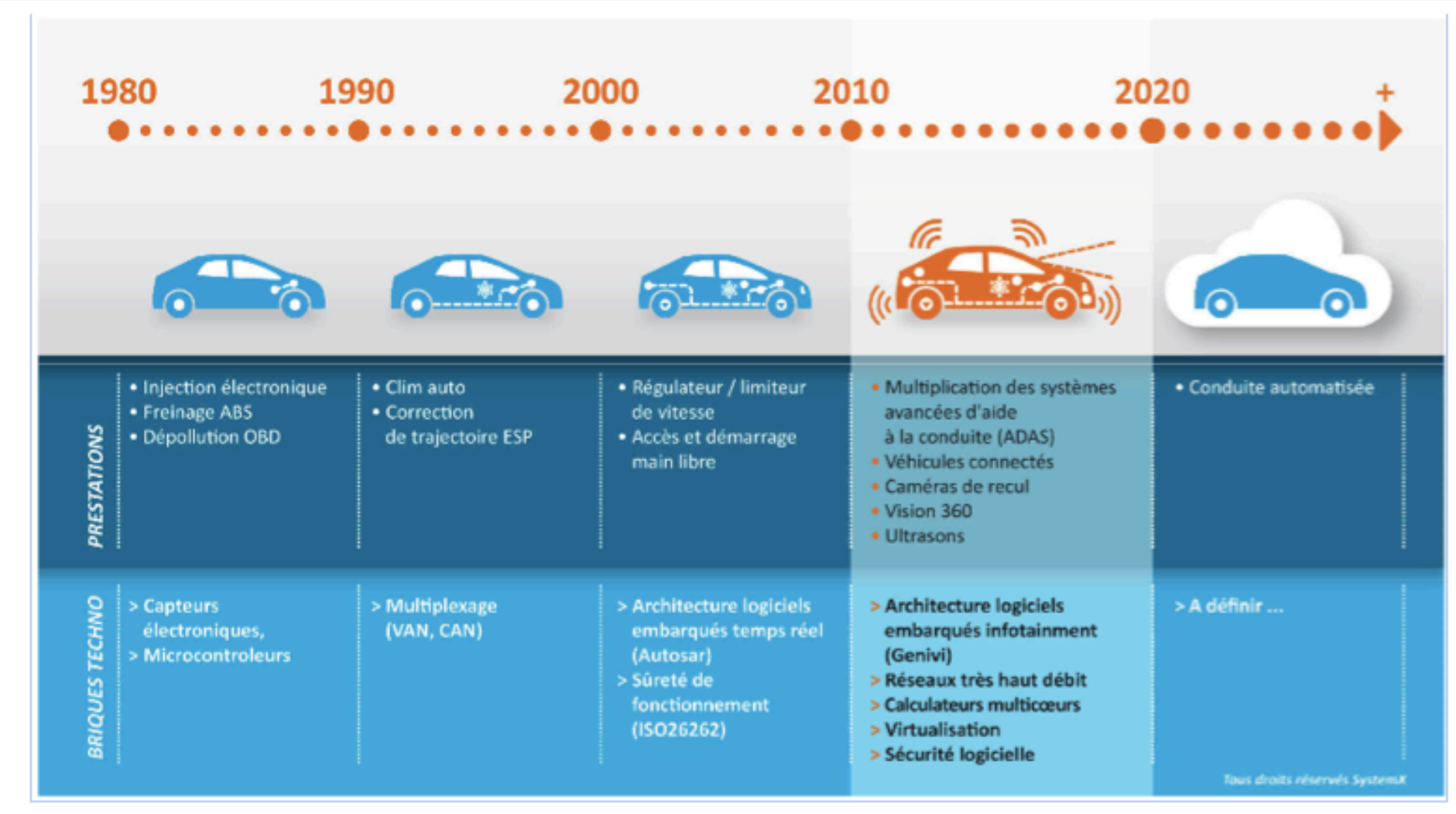
Evolution des systèmes informatiques

depuis 1990 ...

- Démocratisation des ordinateurs personnels de bureau
- Développement des consoles de jeux vidéos
- Bulle Internet
- Multiplication des systèmes d'exploitations et des langages de programmation
- Smartphone, tablettes, Internet des objets, objets connectés

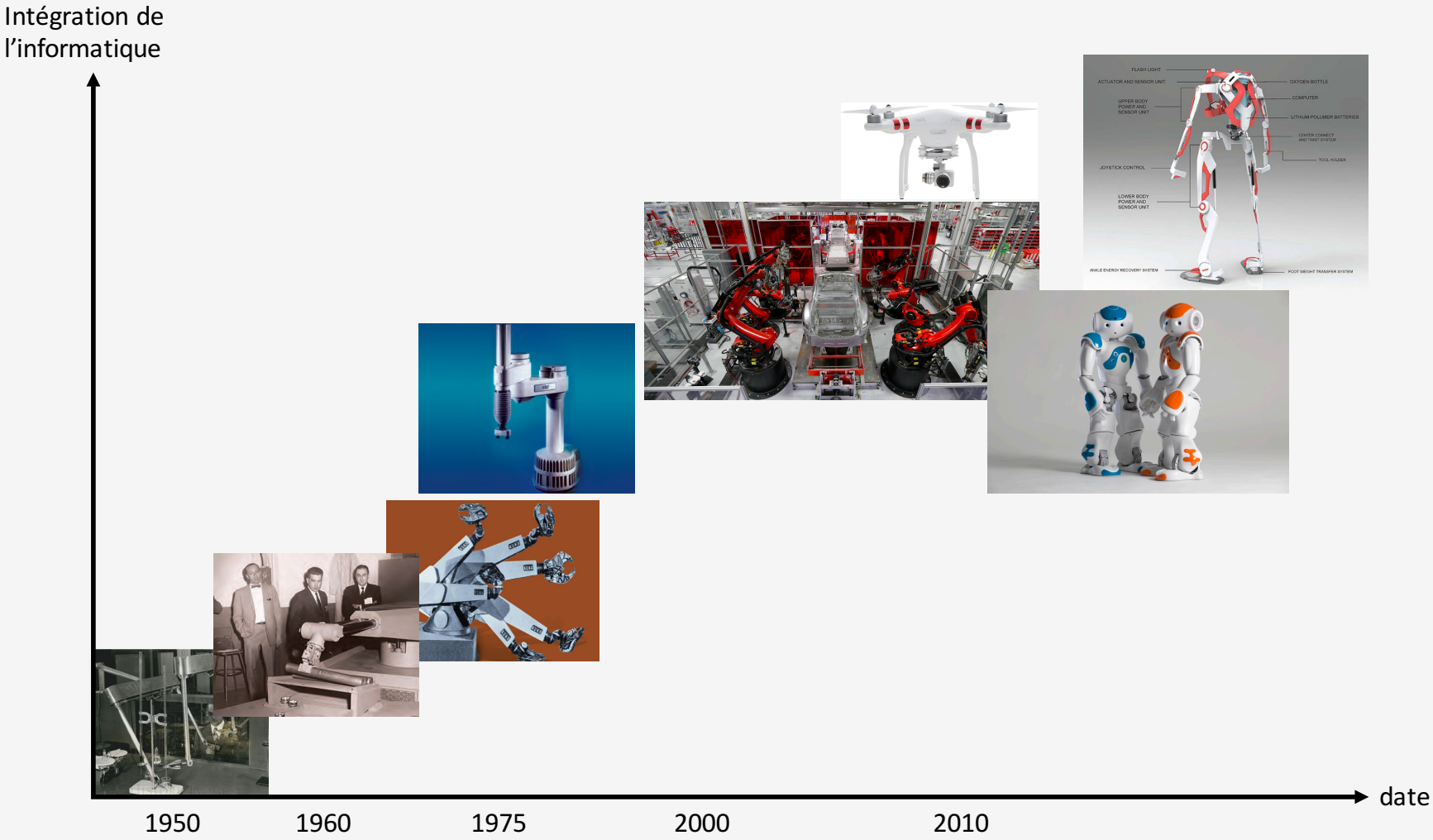
La miniaturisation et l'integration des composants ont fait apparaître les nouvelles générations de PC *nano-ordinateurs* : Raspberry Pi, Beaglebone, Arduino, etc. Les systèmes embarqués sont de plus en plus présents dans l'industrie et notre vie quotidienne.

Domaines d'applications : Automobile

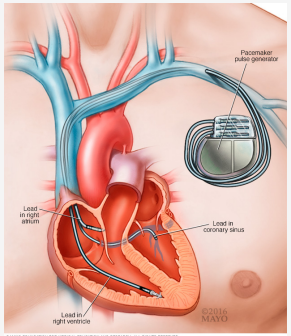


source : L'embarqué

Domaines d'applications : Robotique



Domaines d'applications : Divers



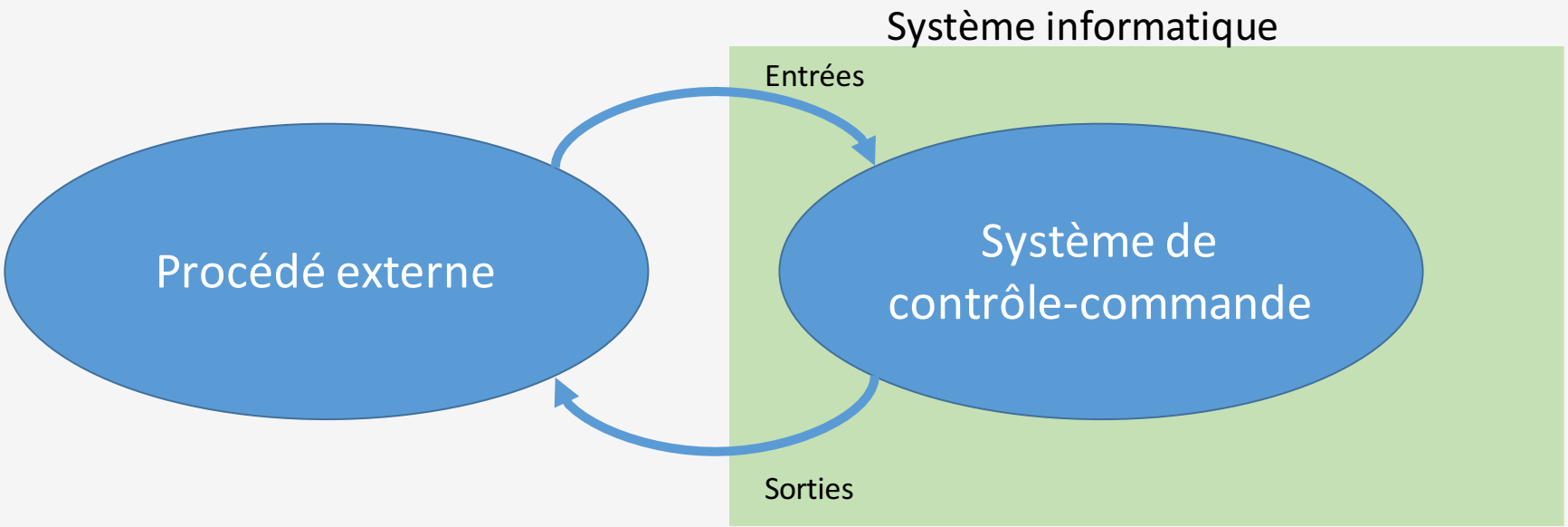
La criticité du système embarqué et de l'informatique varie en fonction de l'application

Classification des systèmes informatiques

- Les systèmes transformationnels
  - résultats calculés à partir de données disponibles à l'initialisation du programme ;
  - les instants de productions des résultats ne sont pas contraints ;
  - applications : calcul scientifique, gestion de base de données
- Les systèmes interactifs
  - résultats dépendent des données produites par le procédé;
  - les instants de productions des résultats ne sont pas contraints ;
  - applications : outils de bureautique
- Les systèmes réactifs
  - résultats obtenus à partir de données produites par le procédé ;
  - contraintes temporelles induites par la dynamique du procédé ;
  - applications : temps réel, contrôle de processus industriels, etc.

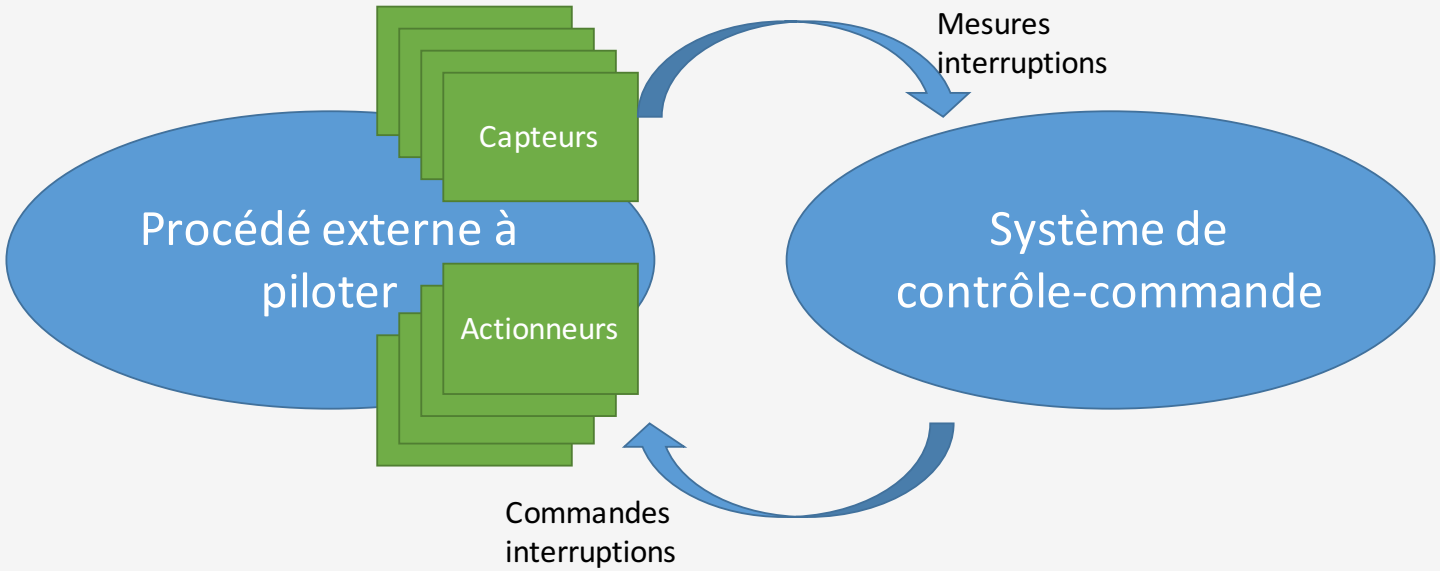
Des systèmes réactifs aux systèmes embarqués

- Terminologie introduite en 1985 par Harel et Pnuelli dans *On the development of reactive systems* ;
- Systèmes en interaction avec leur environnement et réagissent à la vitesse de cet environnement ;



Des systèmes réactifs aux systèmes embarqués

Le système reçoit des informations sur l'état du procédé externe, traite ces données et, en fonction du résultat, évalue une décision qui agit sur cet environnement extérieur afin d'assurer un état stable.



Qu'est ce qu'un système embarqué ?

- Un système embarqué est un système électronique et informatique destiné à une tâche spécifique intimement liée au procédé dans lequel il est intégré. A ce titre il s'agit d'un système réactif.
- Le terme désigne aussi bien le matériel informatique que le logiciel utilisé.
- On parle également de **système enfoui** et en anglais **embedded system**.

Caractéristiques des systèmes embarqués (1)

- la **fiabilité** : probabilité que le système fonctionne correctement à partir du moment où c'est le cas à l'instant $t = 0$;
- la **maintenabilité** : probabilité que le système fonctionne correctement sur un laps de temps d après une défaillance ;
- la **disponiblité** : probabilité que le système fonctionne à un instant t ;
- la **sûreté** : le système ne cause aucun dommage ;
- la **sécurité** : confidentialité et l'authenticité des informations.

Caractéristiques des systèmes embarqués (2)

- Un système embarqué doit être efficace d'un point de vue :
  - **énergétique** ;
  - **logiciel** ;
  - **intégration** (encombrement);
  - **autonomie** (exécution);
  - **coût**.
- Un système embarqué est destiné à une tâche : à prendre en compte dès la conception du système (matériel et logiciel)
- Un système embarqué doit avoir une interface utilisateur dédiée (pas de clavier, pas d'écran, etc.)

Caractéristiques des systèmes embarqués (3)

Un système embarqué :

- respecte des **contraintes temporelles** ;
- restitue une réponse déterministe ;
- est **connecté a un environnement réel** par l'intermédiaire de capteurs et d'actionneurs ;
- est un **système hybride** (combinaison de signaux analogiques et de signaux numériques) ;
- est un **système réactif**.

Bref historique des systèmes embarqués

- début des années 60 : Ordinateur de bord des vaisseaux spatiaux du programme Apollo ;
- 1962 : D-17 d'Autonetics, système de contrôle des missiles nucléaires américains ;
- 1967 : Apollo Guidance Computer (1000 CI $\rightarrow$ NAND) ;
- 1971 : Intel 4004, premier microprocesseur reprogrammable ;
- 1972 : Intel 8008, premier microprocesseur 8 bits ;
- 1974 : Intel 8080, microprocesseur 8 bits, succès grand public ;
- 1979 : création du MC68000, microprocesseur 16/32 bits ;
- Après ... tout s'accélère

Les domaines d'applications

1. **Calcul généraliste** : applications identiques à celles des PC de bureau mais embarquées (tablettes, téléphone portable,etc.). Consoles de jeux vidéos, etc.
2. **Contrôle de systèmes** : motorisation, voitures, avions, industrie chimique, navigation, robotique, etc.
3. **Traitement du signal** : compression vidéos, radar, etc.
4. **Réseaux et communication** : internet, transmission de données, commutation, téléphonie, etc.

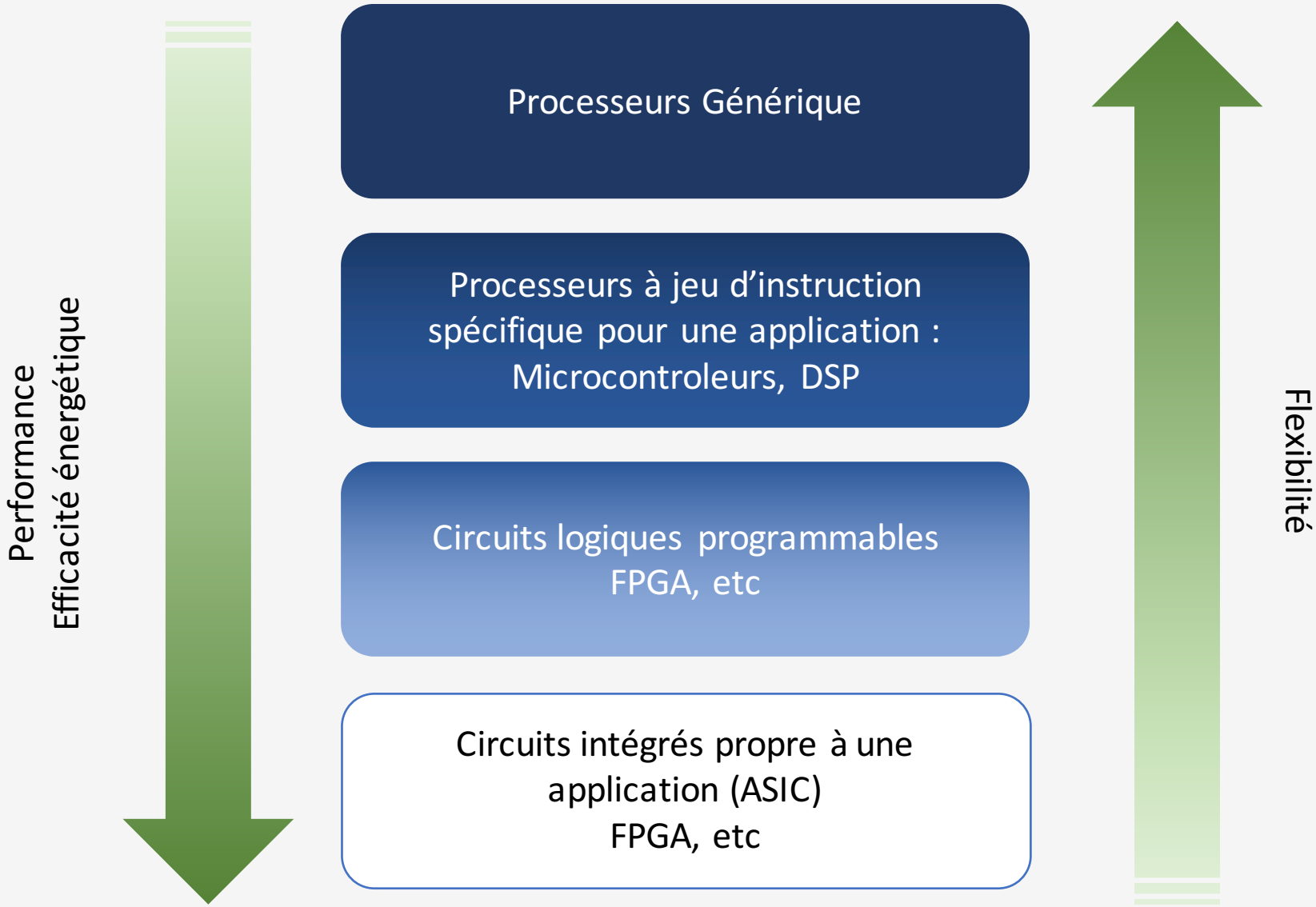
Où trouve-t'on les systèmes embarqués ?

- Installations industrielles : capteurs intelligents, actionneurs connectés, IoT, etc.
- Automobile : régulateurs de vitesses, calculateur, IHM, etc.
- Avionique : pilote automatique, calculateur interne, etc.
- Aérospatiale : fusées, satellites, etc.
- Domestique : téléphone portable, cartes à puces, consoles de jeu, appareils photos, etc.
- Médicale : imageurs, capteurs, pacemakers, etc.
- Militaire : guidage de missile, radars, etc.

Les systèmes embarqués en quelques chiffres

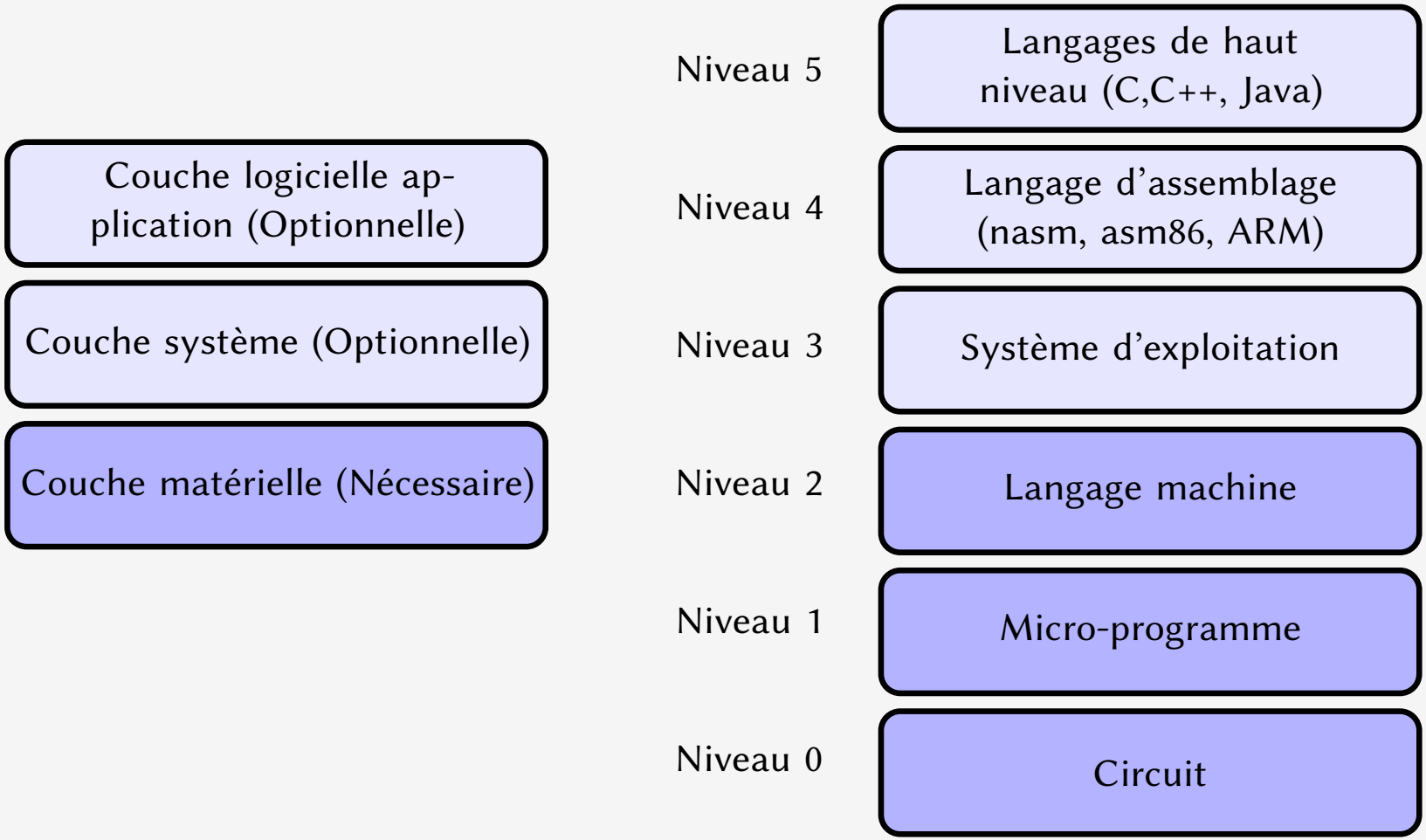
- Les objets connectés ont dopés l'industrie de l'embarqué ;
- En 2008 et 2013, chiffre d'affaire mondial de 113 milliards de dollars ;
- Transparency Market Research (TMR) estime que la marché devrait atteindre 233 milliards de dollars en 2021 (croissance annuelle de 6,4%)
- Industrie 4.0, Internet des objets (IoT), etc...
- Démocratisation des processeurs utilisés dans les systèmes embarqués

Diversité des systèmes embarqués

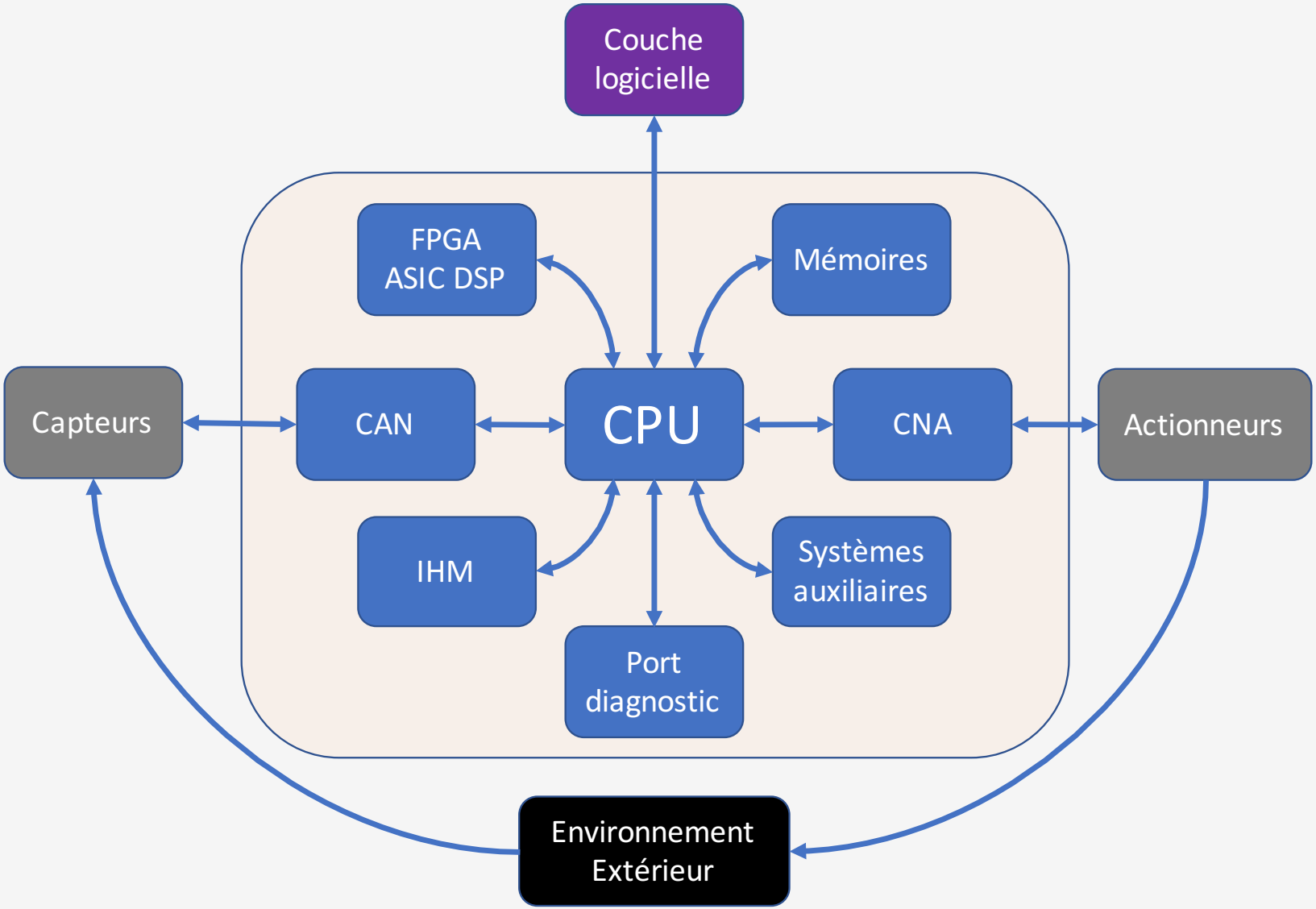


Contents

Détails de l'architecture en couches



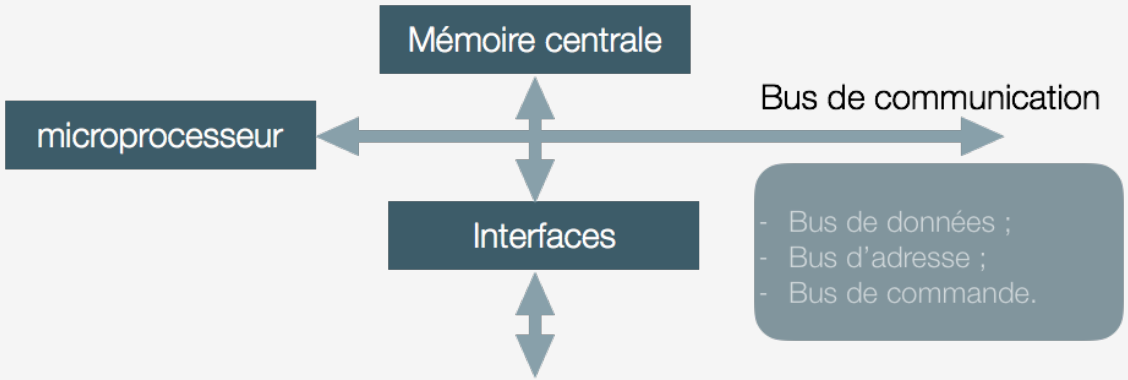
Structure typique d'un système embarqué



Couche matérielle des systèmes embarqués

Architecture minimale

- **Unité centrale de calcul** (Central Processor Unit - CPU) constituée d'un microprocesseur, qui permet au système informatique d'exécuter les programmes, lire les données et restituer des résultats ;
- **Mémoire** - unité physique où les données et/ou programmes sont stockées ;
- **Interface Entrées/Sorties** permettent de communiquer avec l'extérieur/périphériques.
- **Bus** - interconnexion des différents composants



Le microprocesseur

Définition et caractérisitiques

- Exécute une série d'instruction appelée **programme** ;
- Les instructions et les données = en mots binaires (**code machine**) ;
- **jeu d'instruction** et **architecture interne** ;
- Composants spécialisés pour le fonctionnement (mémoire, bus, etc.)

Différences avec un microcontrôleur

Frontière entre μcontrôleur et μprocesseur floue. Les différences entre les deux sont :

- Toutes les fonctions nécessaires à l'utilisation pour une fonction donnée sont intégrés sur la puce ;
- Consommation électrique réduite et coût moins élevé.

Le microprocesseur : du programme au code machine

Langage machine ou code machine

C'est une suite de bits (langage binaire) interprétée par le processeur d'un ordinateur lors de l'exécution d'un programme.

- le langage natif du processeur $\Rightarrow$ seul qui soit reconnu par celui-ci ;
- chaque processeur possède sont propre langage machine ;
- Si un processeur A est capable d'exécuter toutes les instructions d'un processeur B alors on dit que A est compatible avec B. L'inverse n'est pas forcement vrai, A peut avoir des instructions supplémentaires que B ne connaît pas ;
- Il est généré généralement par un **compilateur** d'un langage de programmation ou par l'intermédiaire d'un **bytecode**.

Le microprocesseur : du programme au code machine

Bytecode

Le bytecode est un code intermédiaire entre les instructions machines et le code source. Il est assez proche du code machine à la différence qu'il s'agit d'un interprèteur ou une machine virtuelle qui interprète de code machine

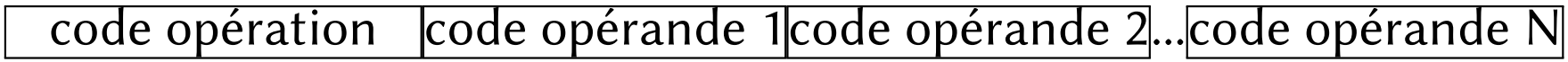
Compilateur

- Un compilateur n'est autre qu'un programme qui est chargé de traduire un programme écrit dans un langage dans un autre langage.
- Le plus souvent le compilateur traduit un langage dit de haut niveau (C, C++, java) en un langage cible (le langage machine) directement exécutable par le processeur.

Le microprocesseur : du programme au code machine

Qu'est ce qu'une instruction ?

C'est l'opération élémentaire qui peut être exécutée par le processeur. Elle est composée de deux champs :



- code opération : action accomplie par le processeur ;
- code opérande : paramètres de l'action (donnée ou adresse mémoire).

Catégories d'instruction : Accès à la mémoire, Opérations arithmétiques, Opérations logiques, Contrôle.

Le microprocesseur : Jeu d'instructions

Jeu d'instruction

Le jeu d'instructions (ISA Instruction Set Architecture) est l'ensemble des instructions machines qu'un processeur peut exécuter. Il s'agit de l'ensemble des circuits logiques qui sont câblés. Il existe deux grandes familles d'architecture pour optimiser le chemin d'exécution :

1. Architecture RISC : Reduced Instruction Set Computer / Ordinateur à jeu d'instruction réduit
2. Architecture CISC : Complex Instruction Set Computer / Ordinateur à jeu d'instruction complexe

Le microprocesseur : Jeu d'instructions

Architecture CISC

Pour augmenter les performances des processeurs

- Jeu d'instruction étendu avec mode d'adressage complexe ;
- 1 instruction peut effectuer plusieurs opérations élémentaires ;
- Instructions proches des langages haut niveau.

Avantages

Moins d'instructions = code compact
Micro-programmation : correction du jeu d'instruction
Instructions complexes et très rapides.

Inconvénients

Surface de la puce plus grande = plus de consommation
Compilateur complexe = pas optimal ;

Le microprocesseur : Jeu d'instructions

Architecture RISC

- Nombre d'instruction réduit et limitation des modes d'adressage;
- Pas de micro-programme;
- Instructions de taille fixe $\rightarrow$ durée d'exécution identique;
- Instruction lancée à chaque cycle d'horloge $\Rightarrow$ **pipeline**.

Avantages

Architecture moins complexe
Faible consommation énergie
Interruptions plus rapides
Compilateur plus simple et efficace

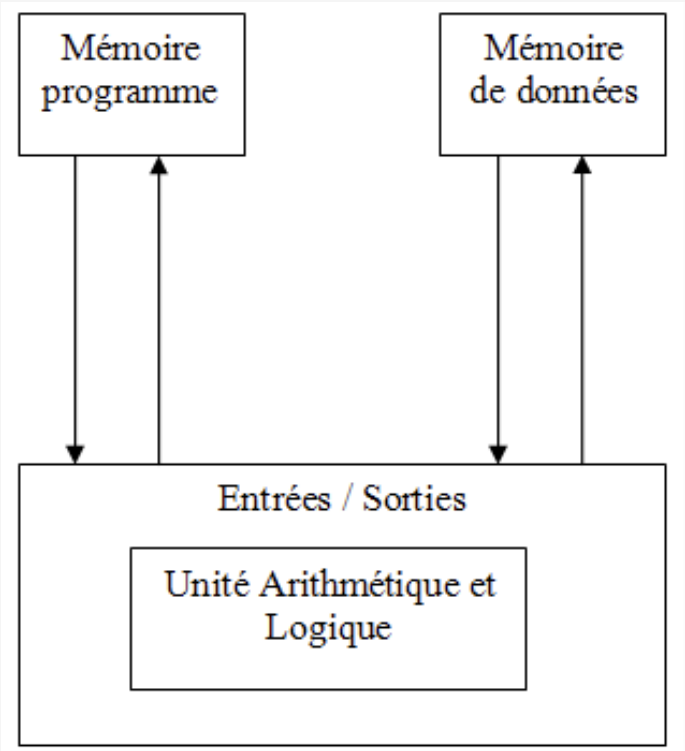
Inconvénients

Taille du programme plus importante
Optimalité de la taille fixe des instructions

Le microprocesseur : Architecture interne

Architecture de Harvard

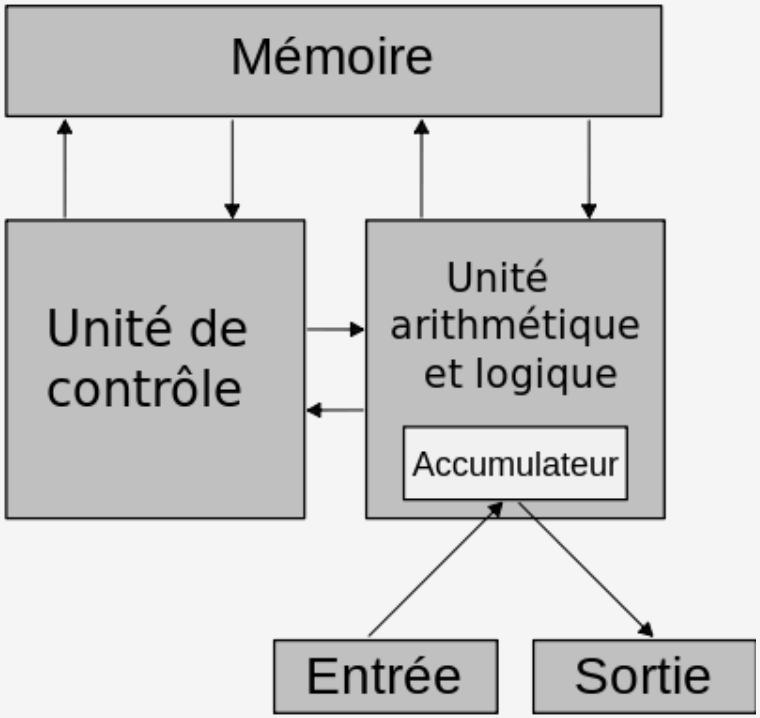
- première mise en œuvre en 1944 sur le système Mark 1.
- rapidité (processeur de type DSP, applications temps-réelles)
- coût/complexité



Le microprocesseur : Architecture interne

Architecture de von Neumann

- première mise en œuvre sur ENIAC.
- quasi-totalité des processeurs actuels
- instructions complexes



Le microprocesseur : Architecture interne

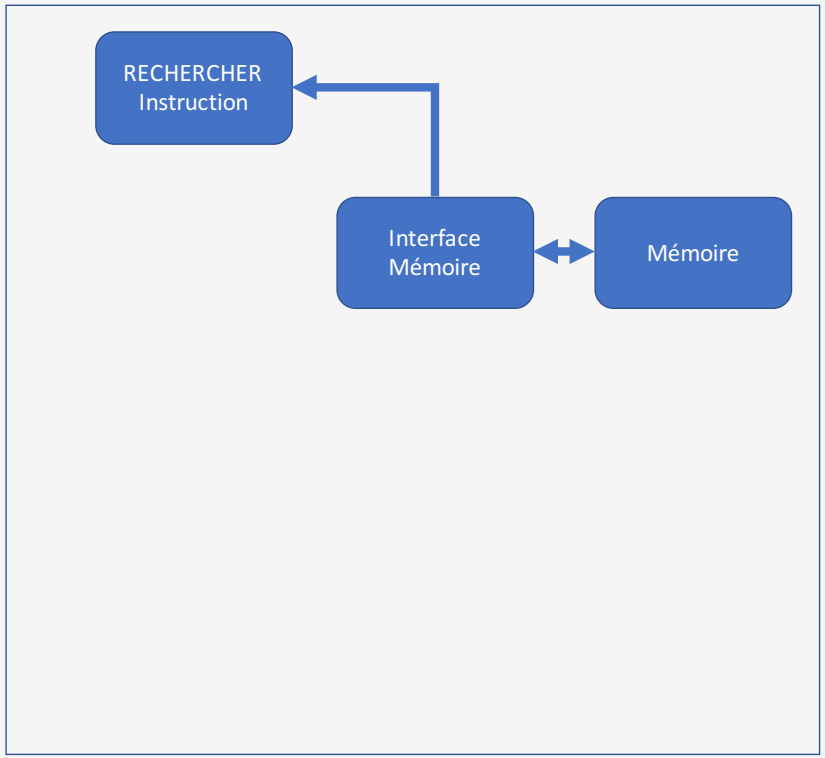
Architecture de von Neumann

- Les programmes et les données sont stockés dans la même zone mémoire.
- Architecture structurée autour de 4 unités :
  - **Unité de calcul (ALU)** : permet d'effectuer les opérations arithmétiques (addition, soustraction, etc.) et logiques (comparaison, opérateurs logiques, etc.) ;
  - **Unité de contrôle (CCU)** : en charge du séquençage des opérations : chercher les instructions et les exécuter. Communique avec ALU pour les opérations arithmétiques et logiques ;
  - **Unité mémoire** : stockage des données manipulées par le micro-processeur et des instructions à exécuter. Mémoire volatile (en cours de traitement) et mémoire permanente (base de la machine) ;
  - **Unité d'entrées-sorties** : interfaçage avec les périphériques externes et l'utilisateur.

Les opérations du microprocesseur

Les opérations décrites ici sont conformes à l'architecture de von Neumann. Le cycle d'exécution est effectué en 4 étapes, que la quasi totalité des microprocesseurs de type von Neumann utilisent :

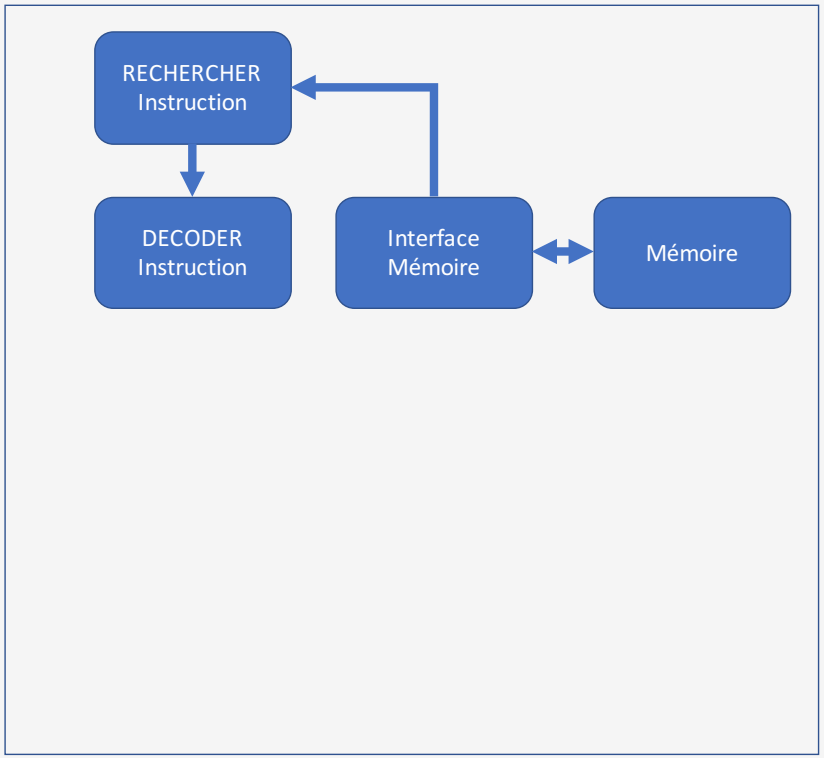
- **fetch** : recherche de l'instruction



Les opérations du microprocesseur

Les opérations décrites ici sont conformes à l'architecture de von Neumann. Le cycle d'exécution est effectué en 4 étapes, que la quasi totalité des microprocesseurs de type von Neumann utilisent :

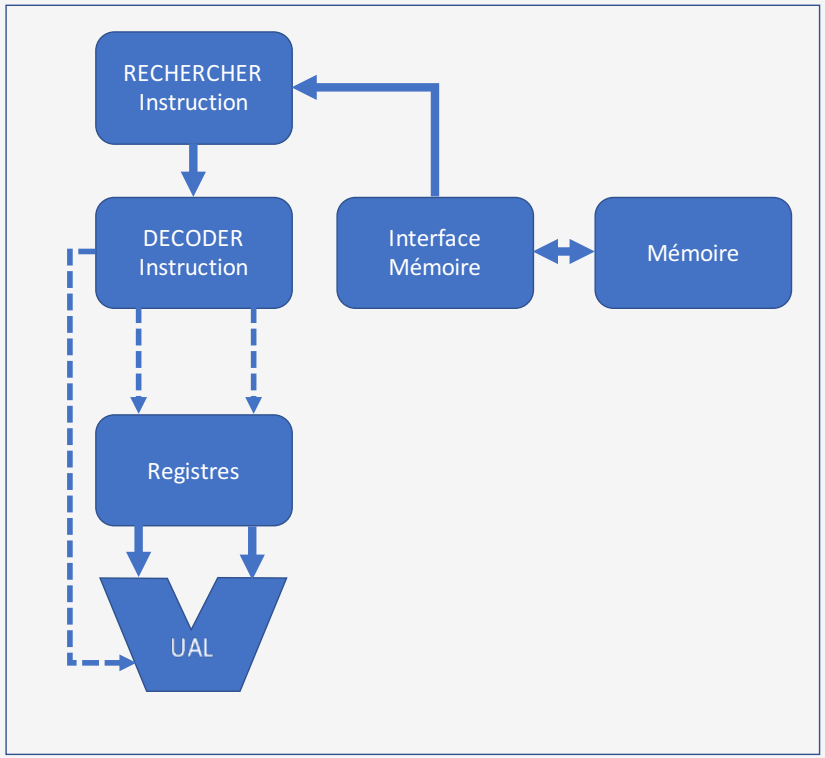
- **fetch** : recherche de l'instruction
- **decode** : interprétation de l'instruction



Les opérations du microprocesseur

Les opérations décrites ici sont conformes à l'architecture de von Neumann. Le cycle d'exécution est effectué en 4 étapes, que la quasi totalité des microprocesseurs de type von Neumann utilisent :

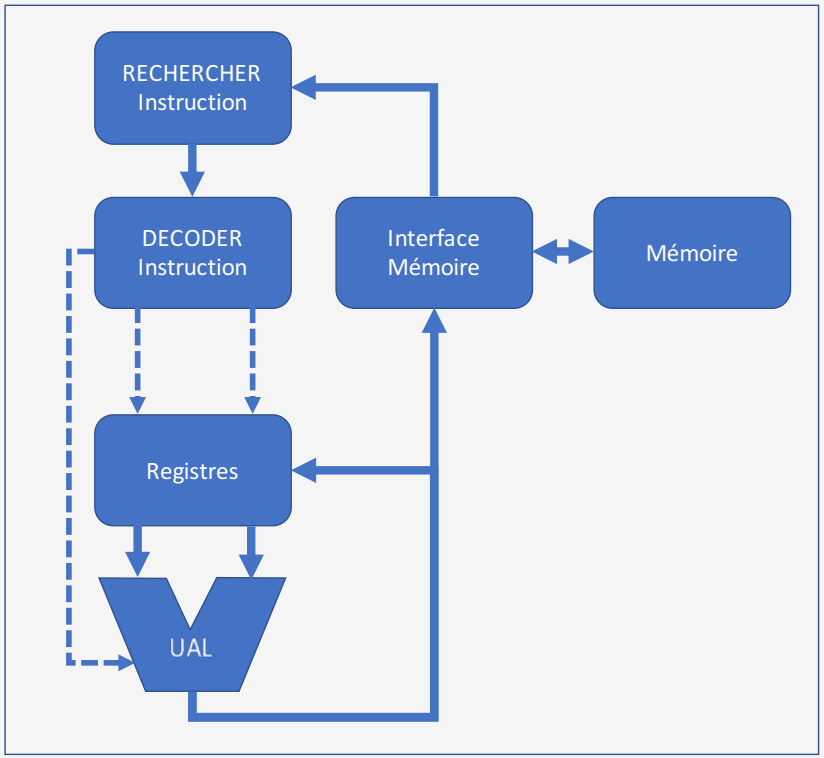
- **fetch** : recherche de l'instruction
- **decode** : interprétation de l'instruction
- **execute** : exécution de l'instruction



Les opérations du microprocesseur

Les opérations décrites ici sont conformes à l'architecture de von Neumann. Le cycle d'exécution est effectué en 4 étapes, que la quasi totalité des microprocesseurs de type von Neumann utilisent :

- **fetch** : recherche de l'instruction
- **decode** : interprétation de l'instruction
- **execute** : exécution de l'instruction
- **writeback** : écriture du résultat



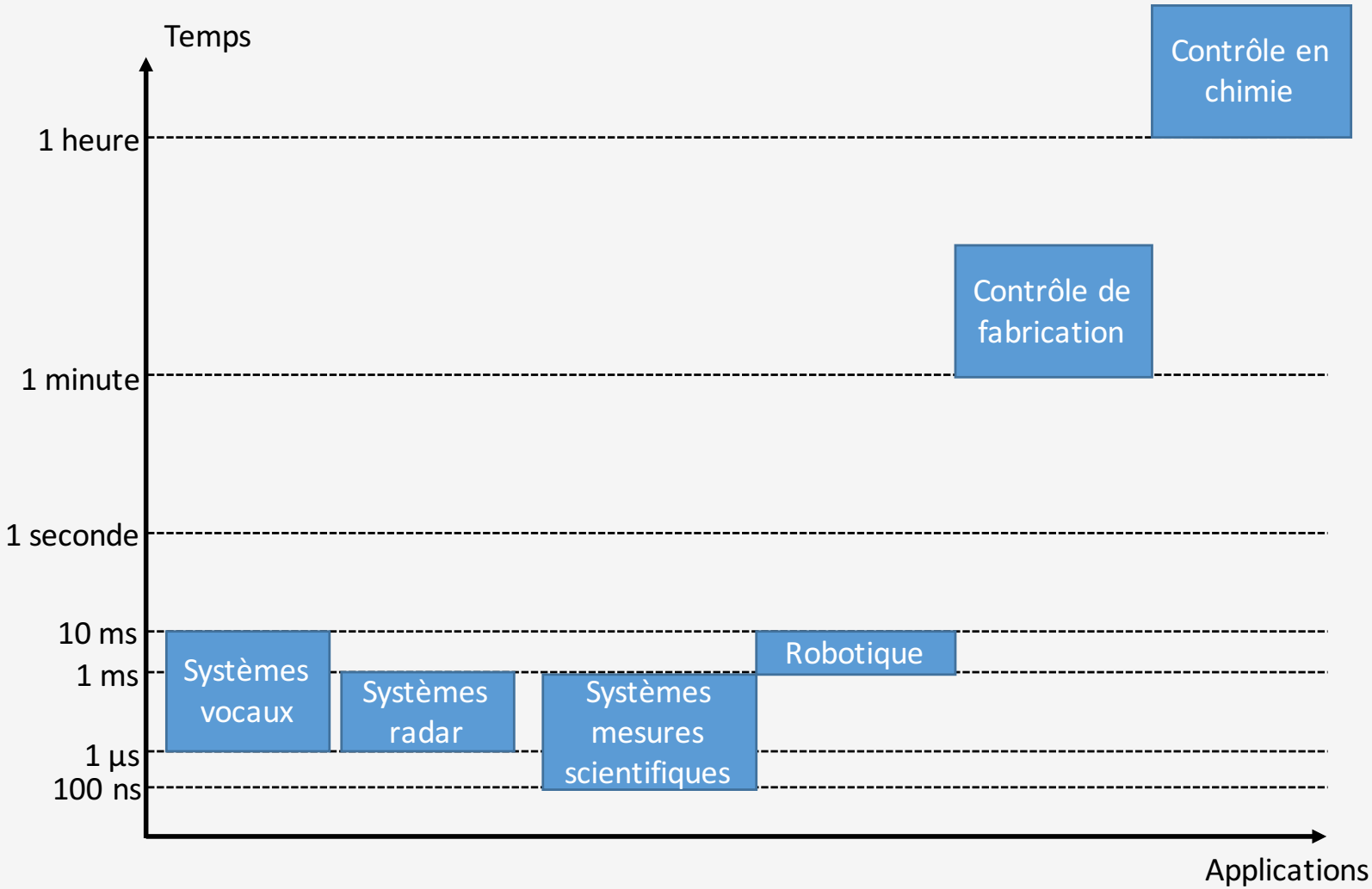
Contents

Systèmes temps réel

Systèmes embarqués et Systèmes temps réel

Un système embarqué est avant tout un système intégrant de l'électronique et de l'informatique pour réaliser une tâche dédiée d'un système plus large ou une machine. C'est également une combinaison de logiciel et matériel, avec des capacités fixées ou programmables.
Un système embarqué est contraint temporellement par la tâche pour lequel il est destiné $\rightarrow$ système temps réel.

Systèmes temps réel



Source : Collet 2005

Systèmes temps réel

Notion de temps-réel [Stankovic1998]

En informatique temps-réel, le comportement correct d'un système dépend, non seulement des résultats logiques des traitements, mais aussi du temps auquel les résultats sont produits. Dont les caractéristiques sont :

- Déterminisme logique : les entrées produisent les mêmes résultats ;
- Déterminisme temporel : respect des contraintes temporelles ;
- Fiabilité : robustesse du matériel et fiabilité du logiciel ;
- **Un système temps-réel est un système prédictible**.

L'Erreur

Un système temps réel N'EST PAS un système rapide.

Systèmes temps réel

Couche logicielle pour les systèmes embarqués

La couche logicielle d'un système embarqué va dépendre :

- ressources matérielles : taille mémoire, interface E/S, fréquence CPU, etc. ;
- la flexibilité et les fonctionnalités souhaités : modularité, complexité de la tâche, interface homme-machine, multi-tâche;

En fonction de ces caractéristiques, la partie logicielle du système embarqué sera plus ou moins complexe et présente.

ATTENTION

- logiciel dans un système embarqué = réduction fiabilité du système
- si la partie logicielle est complexe avec gestion des tâches/processus et mémoire, donc il faudra envisager un système d'exploitation.

Les systèmes d'exploitation

Un OS n'est pas obligatoire pour un système embarqué

Pourquoi un OS dans un système embarqué ?

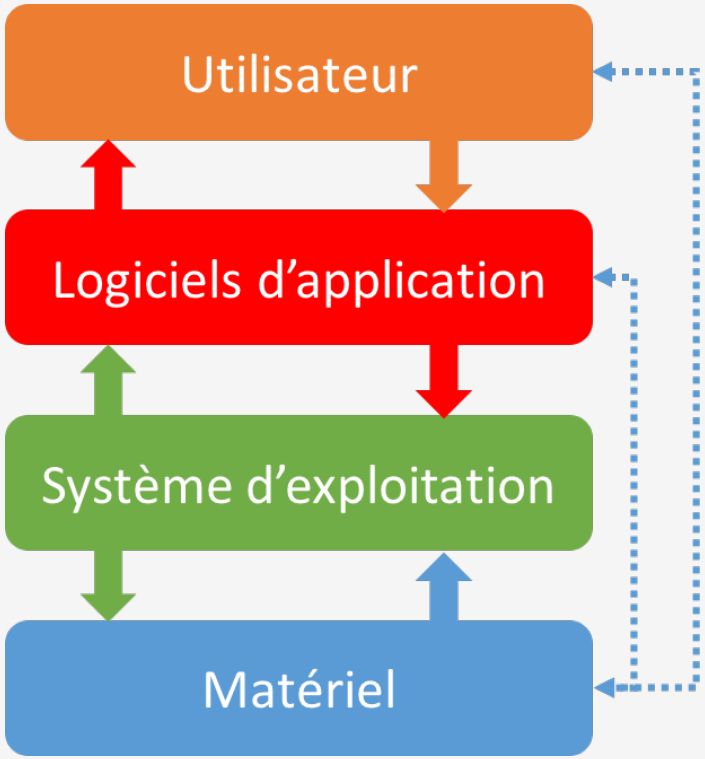
- Affranchir le développeur de bien connaître le matériel → gain de temps de développement ;
- Bénéficier des mêmes avancées technologiques que les applications classiques ;
- Environnement de développement plus performant.

Les systèmes d'exploitation

Définition d'un système d'exploitation (RAPPEL)

Un système d'exploitation (*Operating system OS*) est un ensemble de logiciels qui permettent de :

- contrôler et gérer les ressources matérielles d'un ordinateur ;
- abstraction matérielle pour simplifier l'utilisation ;
- fournir des services aux utilisateurs pour accéder à ces ressources.



Les systèmes d'exploitation

Quelques exemples

- Windows (depuis 1985) c'est aujourd'hui l'OS le plus répandu
- Mac OS (depuis 1984) aujourd'hui Mac OS X version 10.13
- Unix crée en 1969 par Kenneth Thompson
- Linux crée en 1991 par Linux Torvalds (plusieurs distributions sont disponibles open-source)

Les systèmes d'exploitation

Services des systèmes d'exploitation

- Ordonnancement et gestion des tâches ;
- Gestion des interruptions et de la mémoire;
- Communication inter-processus et synchronisation ;
- Entrées/Sorties et pilotes de périphériques ;
- Systèmes de fichiers, protocoles de communication etc.

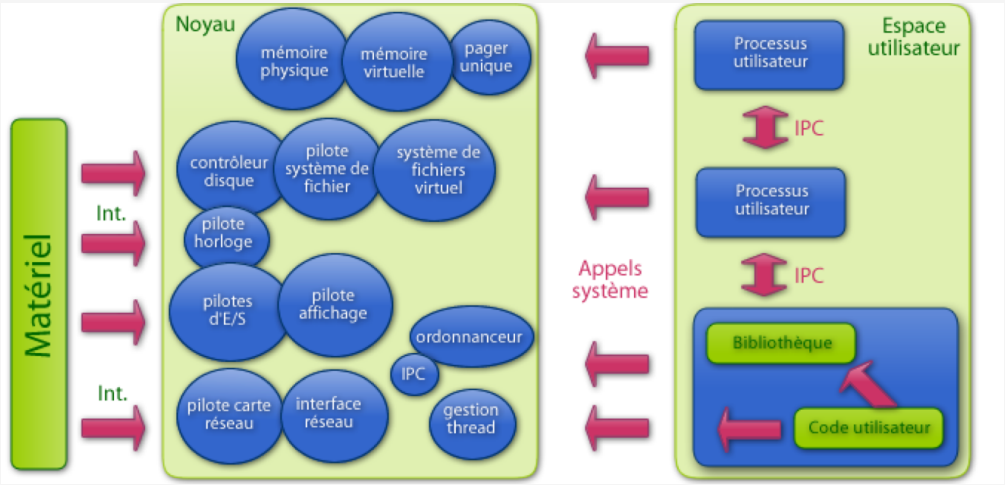
Constituant d'un système d'exploitation

- Le noyau : le cœur du système d'exploitation.
- Gestionnaire de fichier ;
- Appels systèmes : système permettant au noyau de communiquer avec l'extérieur.

Les systèmes d'exploitation

Le noyau

- gestion des périphériques ;
- gestion des **processus** : attribution de mémoire, ordonnancement, synchronisation et communication
- gestion des fichiers ;
- gestion des protocoles ;



Les systèmes d'exploitation

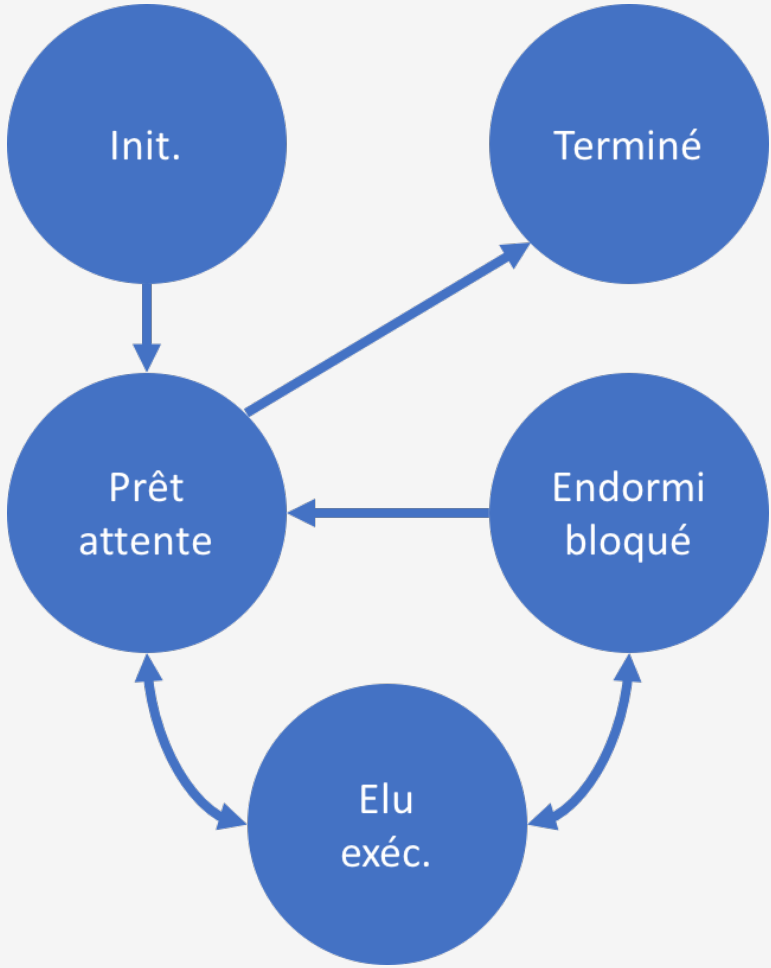
Structure du noyau

- **Noyau monolithique** : tout le code du système d'exploitation est contenu dans le noyau. Les gestionnaires de périphériques ainsi que tout les programmes environnant sont directement intégrés. C'est une structure très programme du matériel, très réactive mais peu flexible, qui génère un code volumineux et donc peu robuste.
- **Noyau monolithique modulaire** : il s'agit d'un noyau monolithique qui permet de charger des modules à la demande (Le noyau Linux est basé sur cette architecture)
- **Micro-noyau** : dans cette structure le noyau est réduit à son stricte minimum, ce qui le rend plus robuste. Les modules sont pour déportés en mode utilisateur, ce qui rend le système tolérant aux fautes. Par exemple Windows fonctionne sur ce principe mais avec un micro noyau enrichi de certaines fonctionnalités très sollicitées.

Les systèmes d'exploitation

Processus

- instance de programme en cours d'exécution sur un ou plusieurs processeurs sous le contrôle d'un système d'exploitation ;
- ensemble d'instructions à exécuter, de données stockées en mémoire ;
- Un processus peut créer un ou plusieurs processus enfant(s) ;
- communication inter-processus et synchronisation ;



Les systèmes d'exploitation

Thread ou processus *léger*

- un **thread** est une séquence d'exécution du code d'un programme au sein d'un processus ;
- plusieurs threads peuvent être exécutés dans un même processus, c'est le **multithreading** ;
- un thread ne peut exister qu'au sein d'un processus lourd (processus) ;
- les ressources d'un processus sont partagées par tous les threads qu'il contient ;
- un thread possède sa propre pile d'exécution, un identificateur et un pointeur d'instruction ;

Les systèmes d'exploitation

Appels systèmes

- fonction primitive fournie par le noyau et utilisée par des programmes s'exécutant dans l'espace utilisateur ;
- généralement ces fonctions sont disponibles dans des bibliothèques ;
- le mécanisme des interruptions logicielles est utilisé pour passer le contrôle au noyau ;
- il existe différentes catégories d'appels systèmes : gestion des processus, gestion des signaux, gestion du système de fichier, gestion des périphériques d'entrées/sorties, etc.

Les systèmes d'exploitation

Ordonnanceur

L'ordonnanceur est un élément clé du noyau d'un système d'exploitation qui va gérer l'ordre d'exécution des processus. Il a plusieurs objectifs :

- Assurer que chaque processus en attente d'exécution reçoive sa part de temps CPU ;
- Minimiser le temps de réponse ;
- Optimiser l'utilisation du CPU ;
- Gérer les priorités entre les tâches ;

Les systèmes d'exploitation

Communication inter-processus IPC

Mécanismes permettant à des processus concurrents de communiquer entre eux :

- la **communication par signaux** est similaire à une interruption logicielle (asynchrone);
- la **communication par tubes** est une connexion unidirectionnelle entre deux processus ;
- la **synchronisation** permet à deux processus d'accéder à une section critique (mécanisme d'exclusion).

Systèmes d'exploitations pour l'embarqué

Les systèmes d'exploitation vs Temps-réel

Les systèmes d'exploitations généralistes ne permettent généralement pas de répondre aux contraintes du temps-réel car :

- Optimisation locale → non prédictible
- Mémoire virtuelle → non prédictible
- Equité temps CPU pour chaque processus → impact temps de réponse
- Nombreux processus de maintenance pour favoriser l'utilisation
- Gestion grossière du temps
- Indépendance des processus → gestion de l'IPC

Les OS ou exécutifs généralistes sont généralement **non préemptibles**.
Introduction de **latence** par l'utilisation de fonctions haut niveau

Systèmes embarqués temps réel

Classification des systèmes temps-réel

- **Temps-réel mou** (soft real-time) :
  - Le respect des contraintes temporelles est importante mais la violation celles-ci ne provoque pas de graves défaillances
  - vidéoconférence, les systèmes multimédias, etc.
- **Temps-réel dur** (hard real-time) :
  - Le non respect des contraintes temporelles provoque de graves défaillances
  - Système de pilotage d'avion, guidage missile, etc.

Systèmes embarqués temps réel

Systèmes d'exploitation temps-réel [Gangloff et Cuvillon 2008]

- Le noyau permet de respecter des contraintes temporelles fortes ;
- Réactivité par rapport aux interruptions matérielles ;
- Création de tâches périodiques avec fluctuation de la période (jitter) minimale ;
- Exemples : VxWorks (micro-noyau temps réel), QNX (micro noyau temps réel), Xenomai, etc.

Systèmes embarqués temps réel

Notion de latence

- La latence est une notion fondamentale dans les systèmes temps réel, elle est liée au temps de réponse du système.
- La latence est la durée qui s'écoule entre l'instant de réception d'une interruption et l'instant de traitement.

Si la latence n'est pas maîtrisée alors il ne peut y avoir de temps réel.

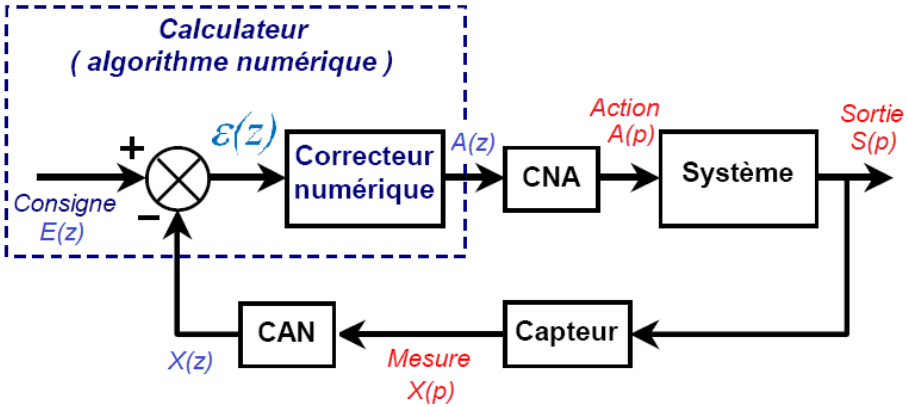
Systèmes embarqués temps réel

Notions fondamentales

- **Tâches préemptibles** : La tâche peut être interrompue à n'importe quel instant et le processeur est alors affecté à une autre tâche. Programmation plus complexe et gestion des accès à des ressources critiques.
- **Tâches non préemptibles** : La tâche ne peut être interrompues qu'à des instants/endroits spécifiques ou à la demande de la tâche elle-même.

Systèmes embarqués temps réel

Les systèmes de contrôle-commande

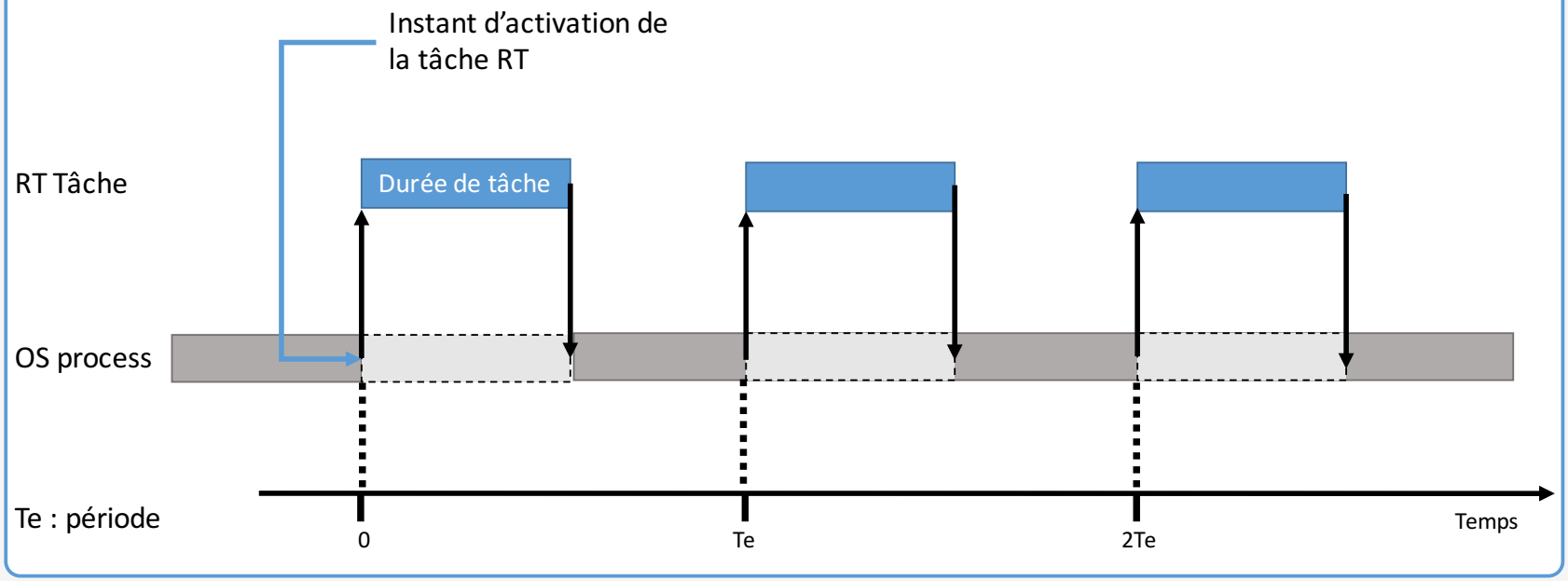


- Boucle d'asservissement cadencée avec une période T_e, à chaque période :
  1. envoyer la commande précédente $A(k-1)$;
  2. acquérir la nouvelle mesure $X(k)$;
  3. comparer avec la consigne $\epsilon(k) = E(k) - X(k)$;
  4. calculer la nouvelle commande $A(k)$;

Il s'agit d'une tâche périodique qui peut être réalisée par un système réactif temps-réel

Systèmes embarqués temps réel

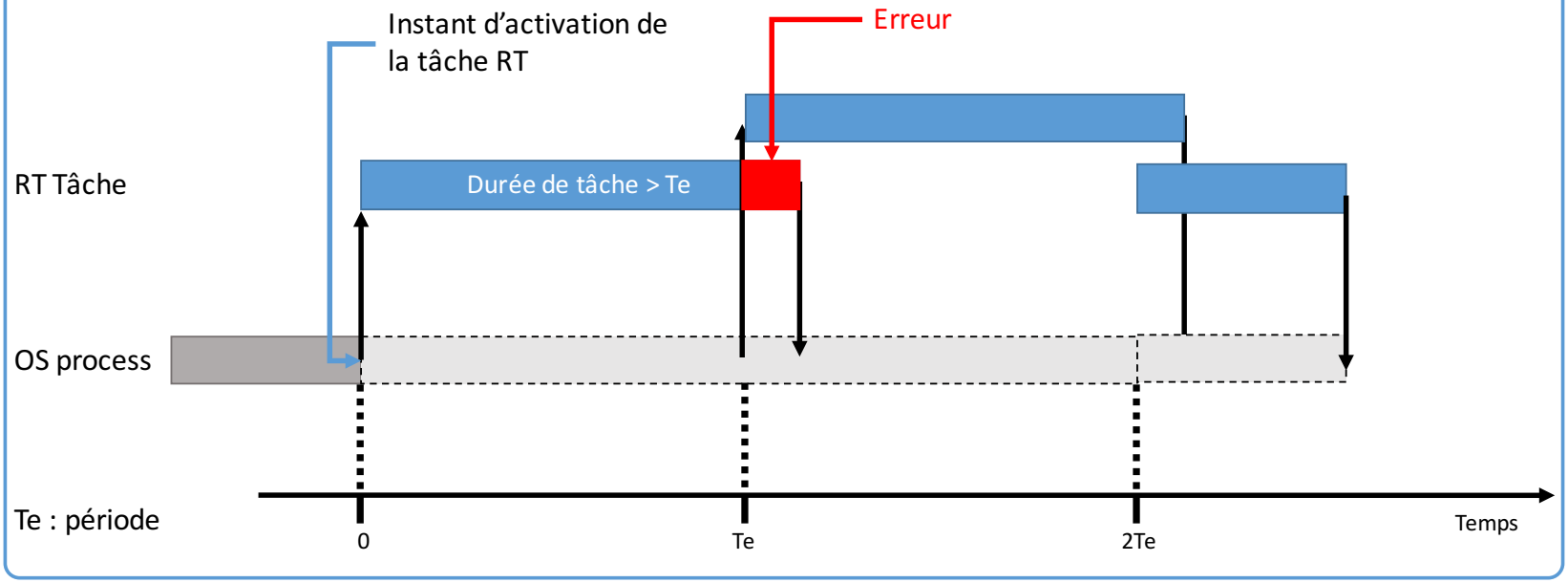
Threads périodiques



Systèmes embarqués temps réel

Thread périodique

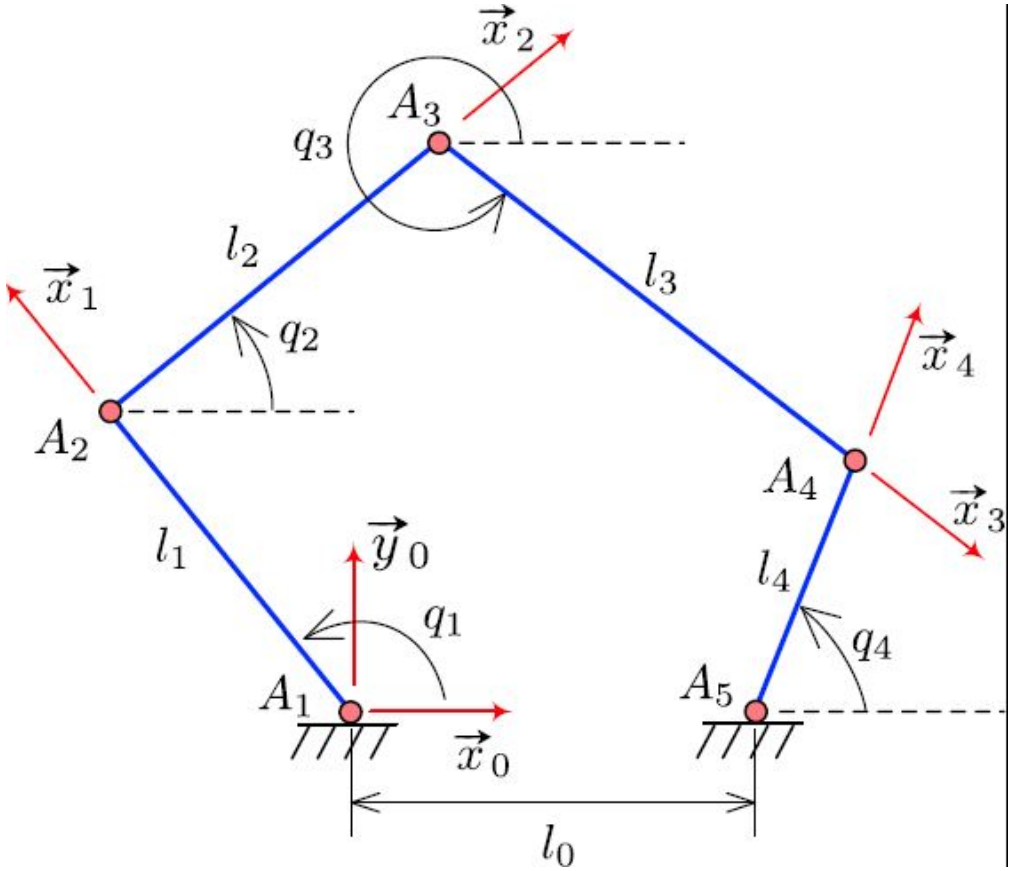
Threads périodiques



Contents

Présentation du système robotique

Système 5 barres



Paramétrage et valeurs numériques

Paramétrage

- $\mathbf{A_0A_5} = l_0\vec{x}_0$
- $\mathbf{A_iA_{i+1}} = l_i\vec{x}_i, \forall\ i\ \in\ [1,5]$
- Coordonnées articulaires : $q_i = (\vec{x}_0, \vec{x}_i)$
- A_1 et A_5 axes motorisés et mesures angulaires q_1 et q_4
- repère de base $\mathrm{R}_0 = \{A_0;\ \vec{x}_0,\ \vec{y}_0\}$
- position de l'organe terminal dans le repère R_0 est $\mathbf{A_0A_3} = x_3\vec{x}_0 + y_3\vec{y}_0$

Valeurs numériques

$l_0 = 137,98\ mm,\ l_1 = l_4 = 80\ mm,\ l_2 = l_3 = 146,3\ mm$

Modélisation du robot

Modèle géométrique direct

Le MGD permet de connaître la situation de l'organe terminal d'un système robotique en fonction de sa configuration. Pour le système 5 barres présenté :

$$[x_3 \ y_3]^T = f\left([q_1 \ q_4]^T\right)$$

Modèle géométrique indirect

Le MGI permet de connaître la ou les configurations d'un système robotique à partir de la situation de son organe. Pour le système 5 barres présenté :

$$[q_1 \ q_4]^T = f^{-1}\left([x_3 \ y_3]^T\right)$$

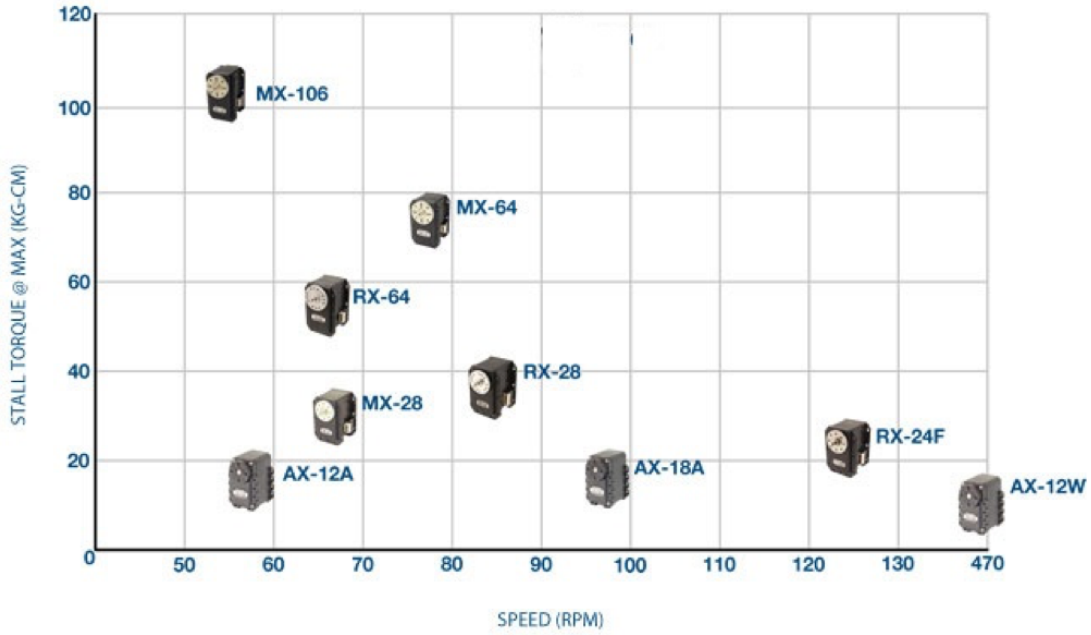
Actionneurs et capteurs

Servomoteurs intelligents : DYNAMIXEL

- Servomoteurs intelligents développés par la société ROBOTIS sont des actionneurs tout en un spécialement conçus pour la robotique.
- Bus de communication numérique de type RS485, TTL, etc.
- Câblage simplifié par Daisy-Chain, identification simple de l'actionneur par ID
- Contrôle en position
- Contrôle de température, réglage du couple moteur
- Etat du servomoteur par indication de LED
- Electronique simplifiée

Actionneurs et capteurs

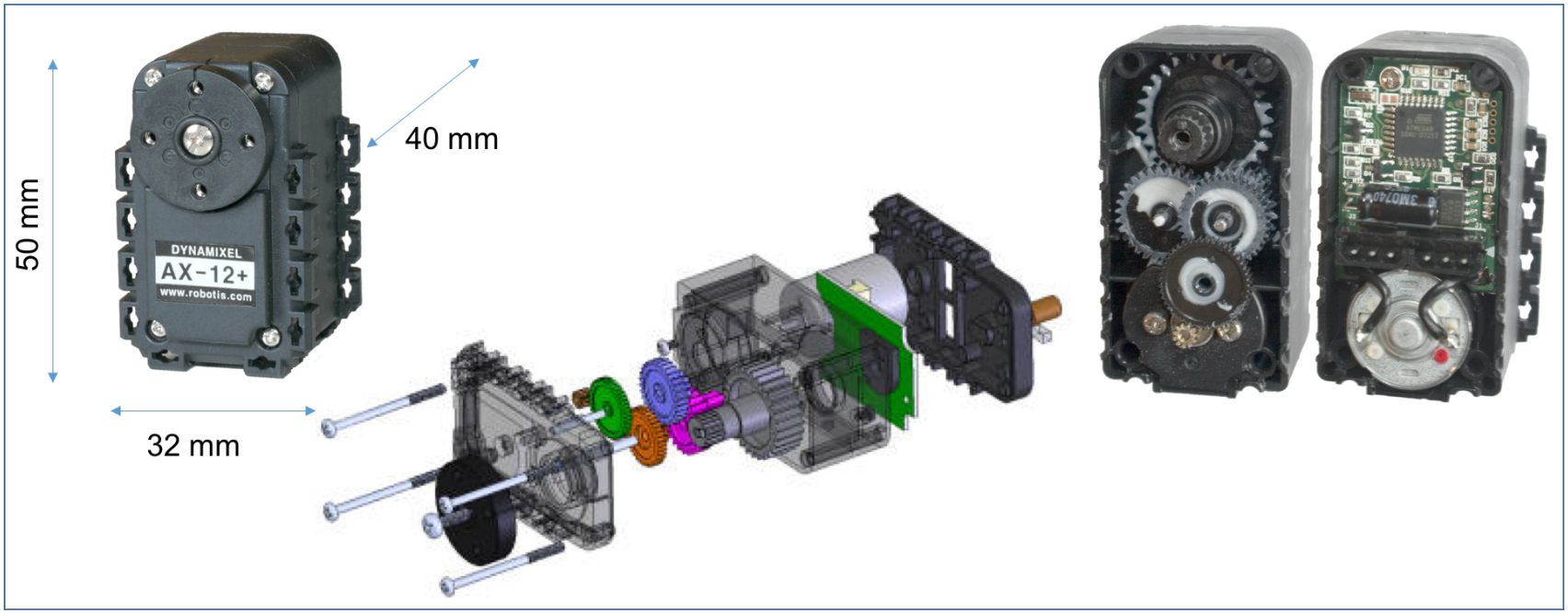
Servomoteurs intelligents : DYNAMIXEL



- Grand public :
  - séries AX, RX et MX
- Intermédiaire :
  - série X
- Professionnelle :
  - série Pro

Actionneurs et capteurs

Servomoteurs intelligents : AX-12



- Poids : 55 g;
- Rapport de réduction : 1:254 ;

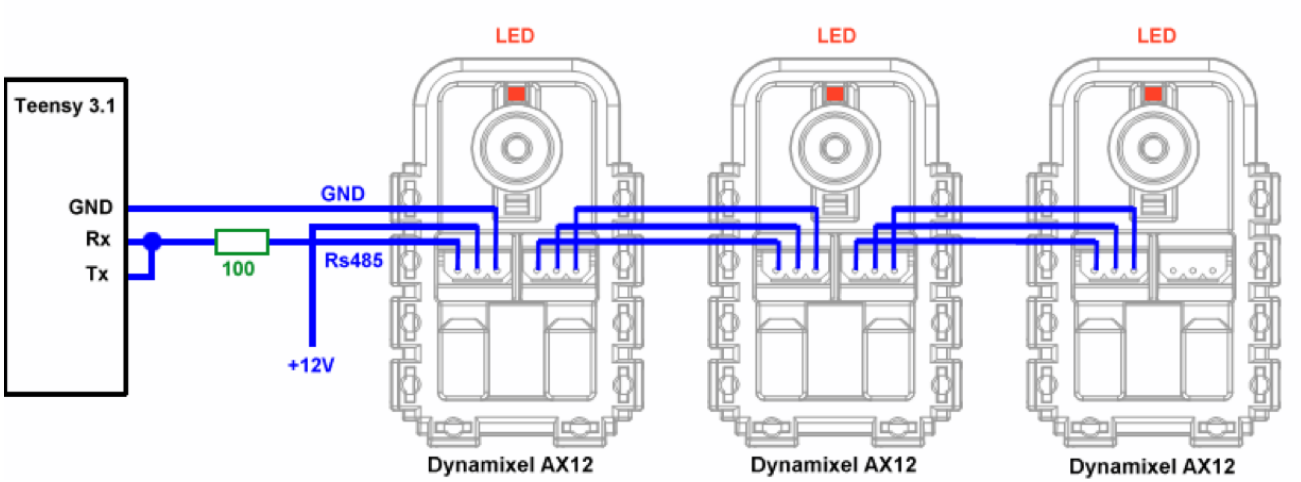
Actionneurs et capteurs

Servomoteurs intelligents : Caractéristiques

- Poids : 55 g, dimensions : 32mm x 50 mm x 40 mm ;
- Résolution : 0,29 ° ;
- Rapport de réduction : 254:1 ;
- Couple de décrochage : 1,52 N.m (12V 1,5A)
- Vitesse sans charge : 59 rpm (12V)
- Protocole : Half Duplex Asynchronous Serial Communication (8bits,1 stop, pas de partié)
- Lien Physique : connecteur de type daisy chain TTL
- Repérage par un identifiant

Actionneurs et capteurs

Servomoteurs intelligents : Communication

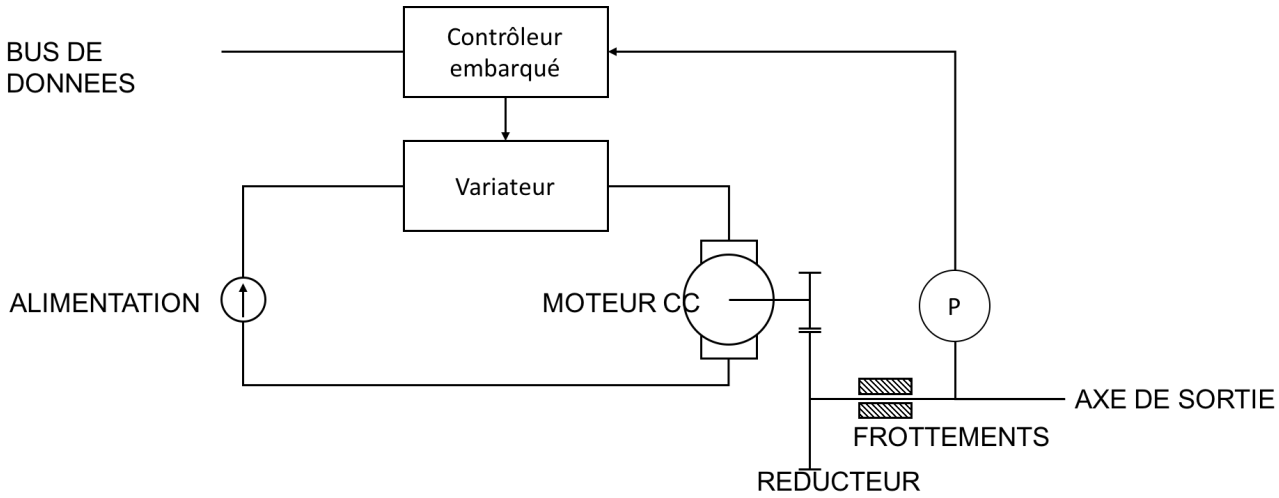


- Chainage des servomoteurs : daisy chain (un connecteur entrée, un connecteur sortie) ;
- 2 types de protocoles de communication :
  - Half Duplex Asynchronous Serial Communication (Daisy chain TTL 3 fils) (gamme AX, MX-T et XL)
  - RS485 (Daisy chain 4 fils) (gamme MX-R, RX-R, DX et EX)

Actionneurs et capteurs

Servomoteurs intelligents : Contrôle

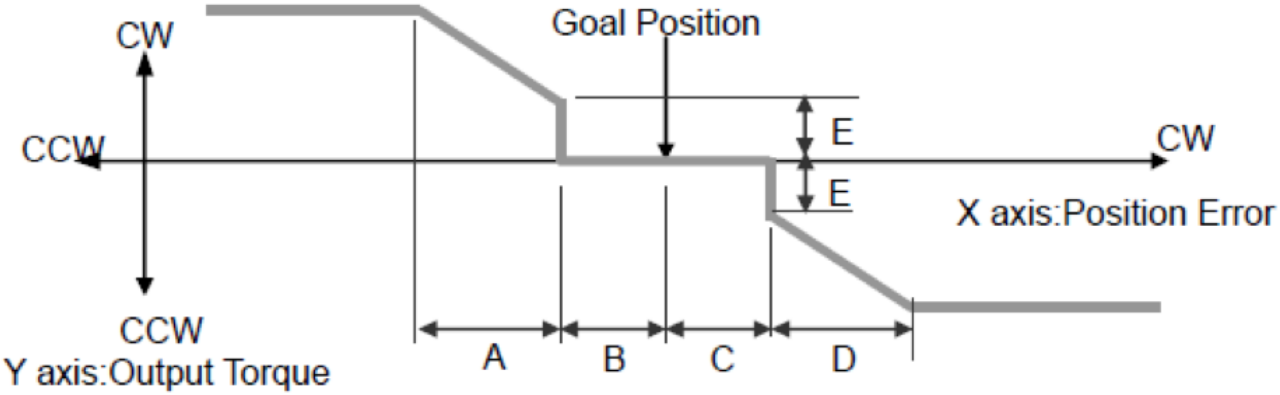
- Communication et contrôle des servomoteurs par l'intermédiaire d'un microcontrôleur ATMega
- Communication numérique des informations : décodage, Conversion numérique analogique vers l'amplificateur pour commander le moteur



Actionneurs et capteurs

Servomoteurs intelligents : Asservissement

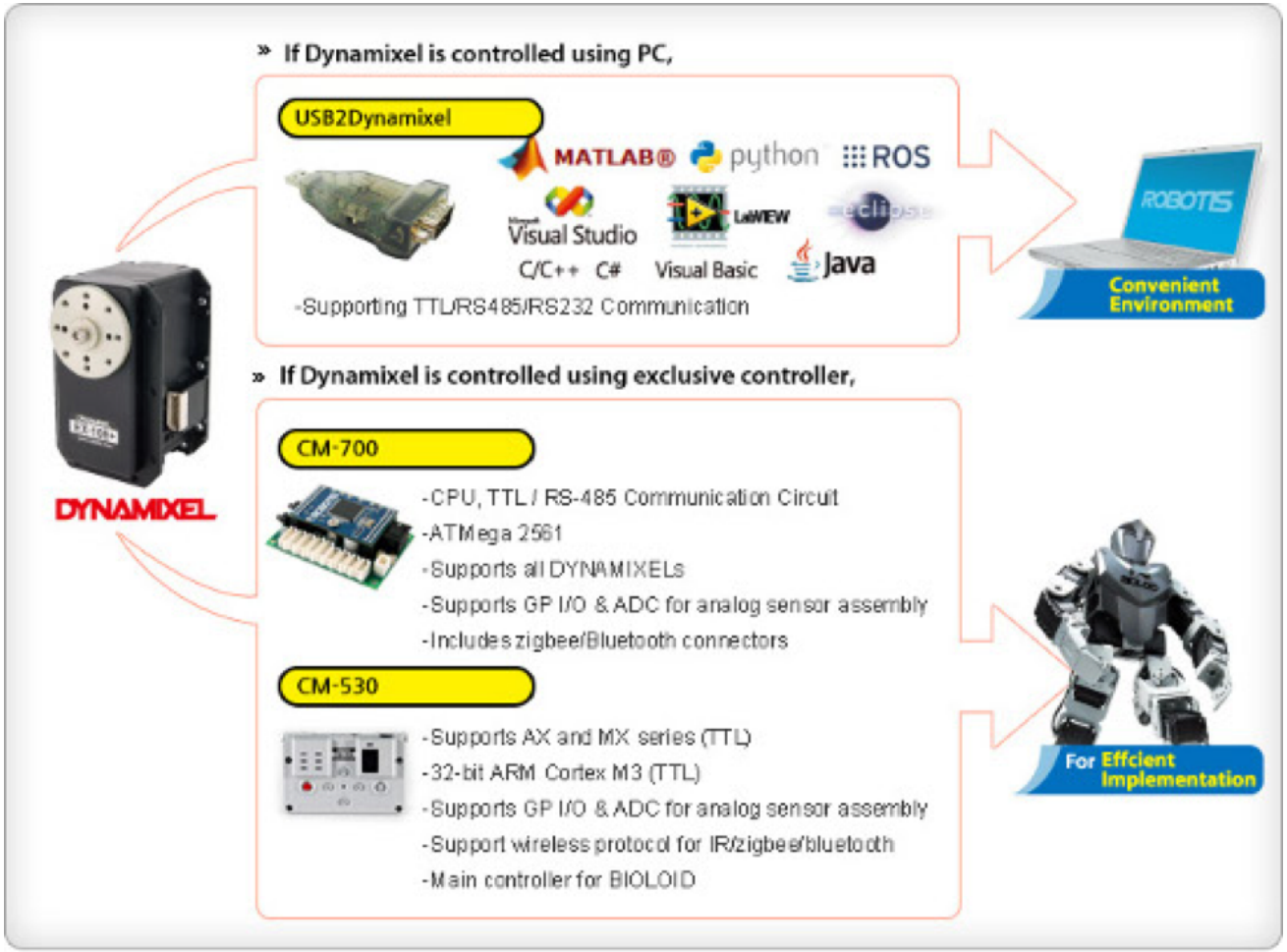
- Asservissement de type proportionnelle avec zone morte



A : CCW Compliance Slope(Address0x1D)
B : CCW Compliance Margin(Address0x1B)
C : CW Compliance Margin(Address0x1A)
D : CW Compliance Slope (Address0x1C)
E : Punch(Address0x30,31)

Actionneurs et capteurs

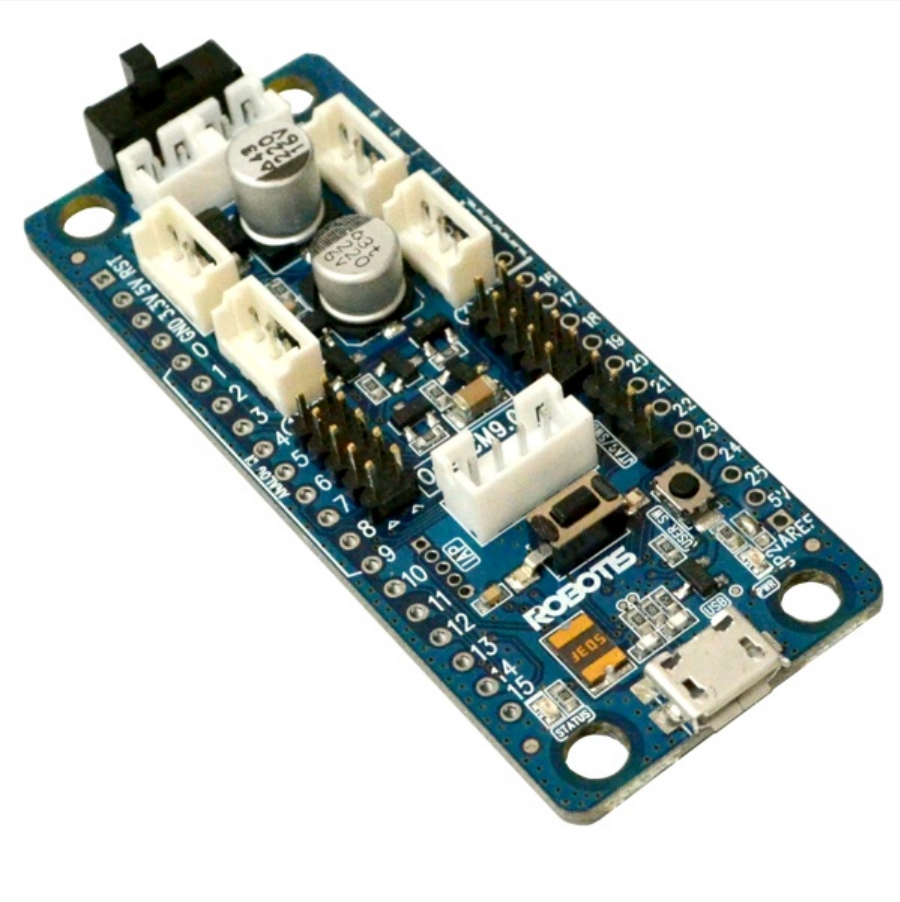
Servomoteurs intelligents : Contrôle



Carte de développement

Carte OpenCM9.04 (ROBOTIS)

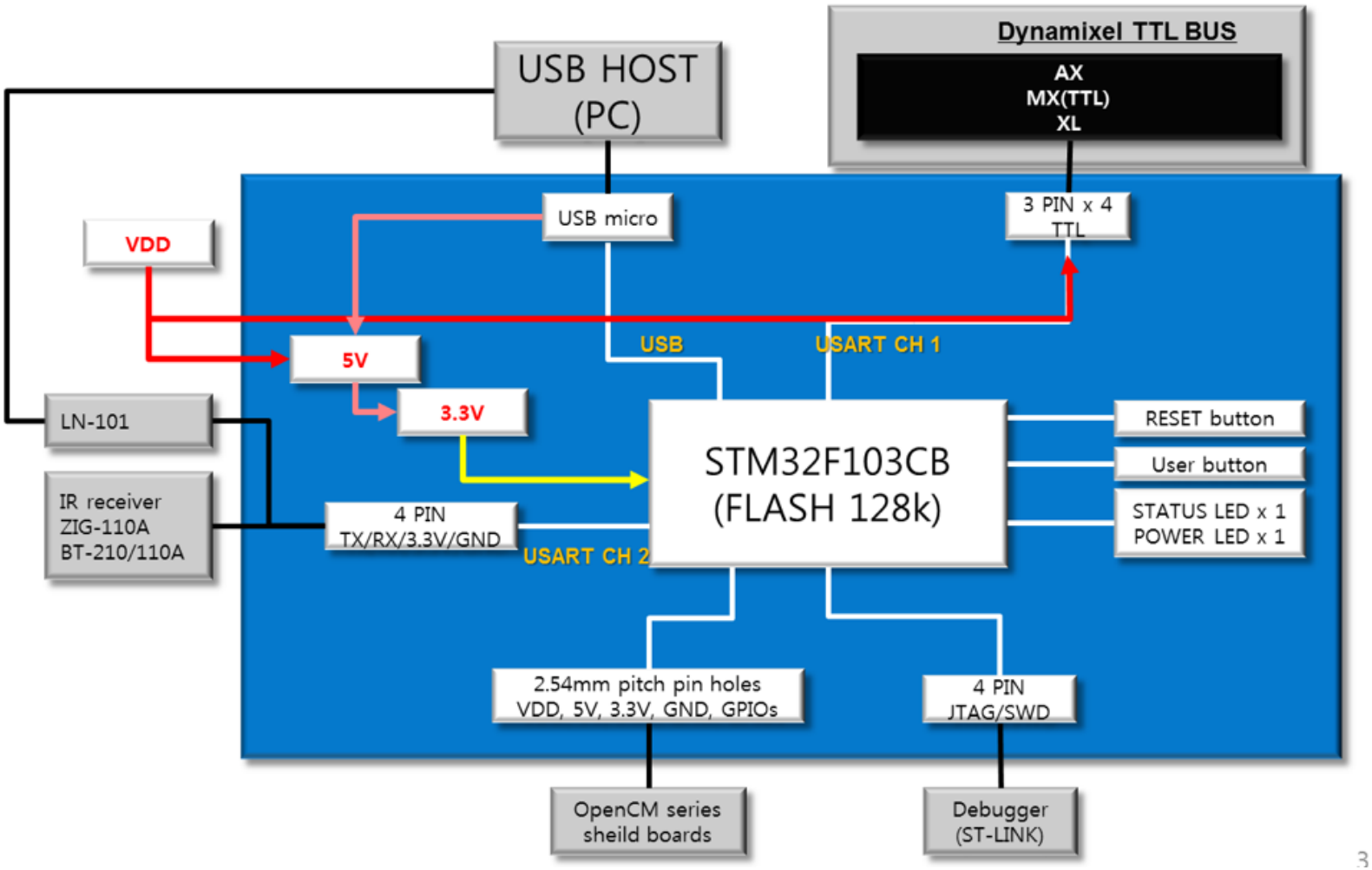
Carte de contrôle open-source pour les actionneurs DYNAMIXEL



- Dimensions 66,5 x 27 mm
- Alimentation par batterie, externe (de 5V à 16V) ou USB
- Communication TTL (3 broches) avec DYNAMIXEL
- CPU : STM32F103CB de STMicroelectronics
- Micro-USB pour charger les programmes sur la carte

Carte de développement

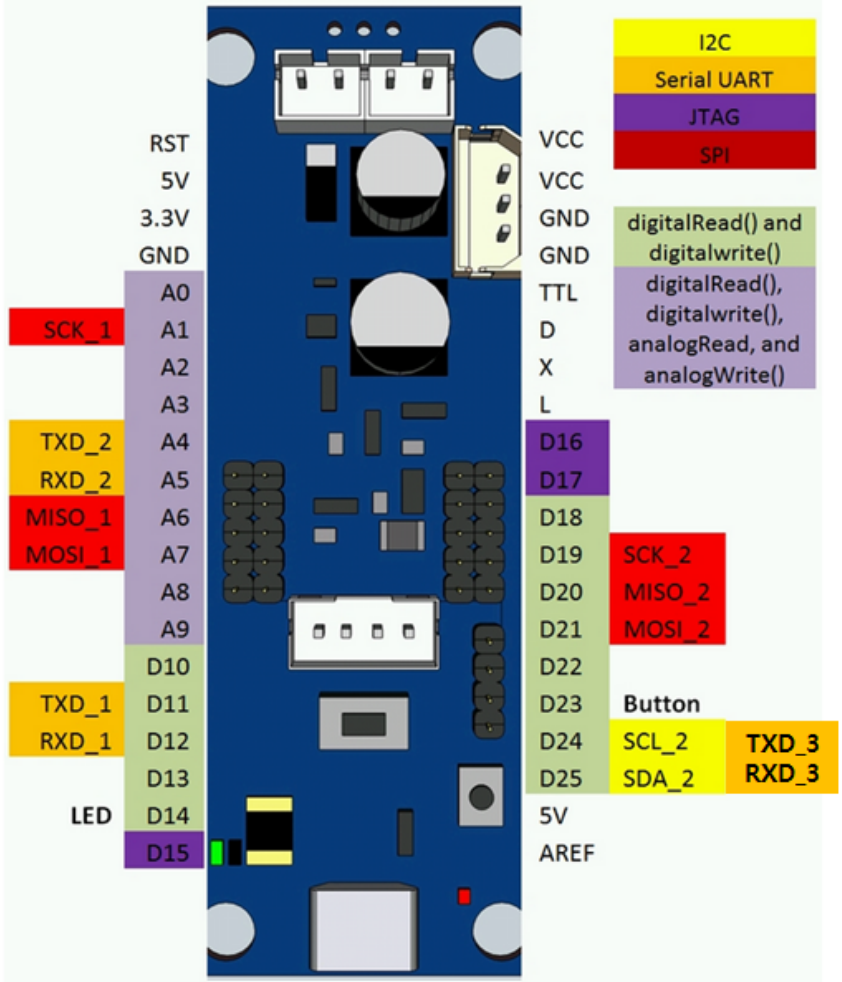
Schéma bloc de la carte



Carte de développement

Caractérisitiques

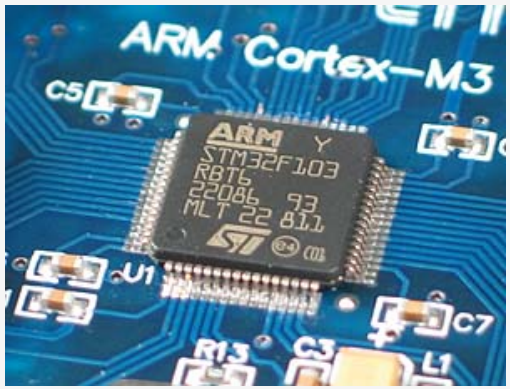
- Entrées/Sorties : 26 GPIO
- Timers : 4 (16bits)
- 10 entrées analogiques 12 bits
- 128 Kb de mémoire FLash
- 20 Kb de SRAM
- horloge : 72 MHz
- USB, USART, SPI, I2C, JTAG, TTL (4 DYNAMIXEL)



Carte de développement

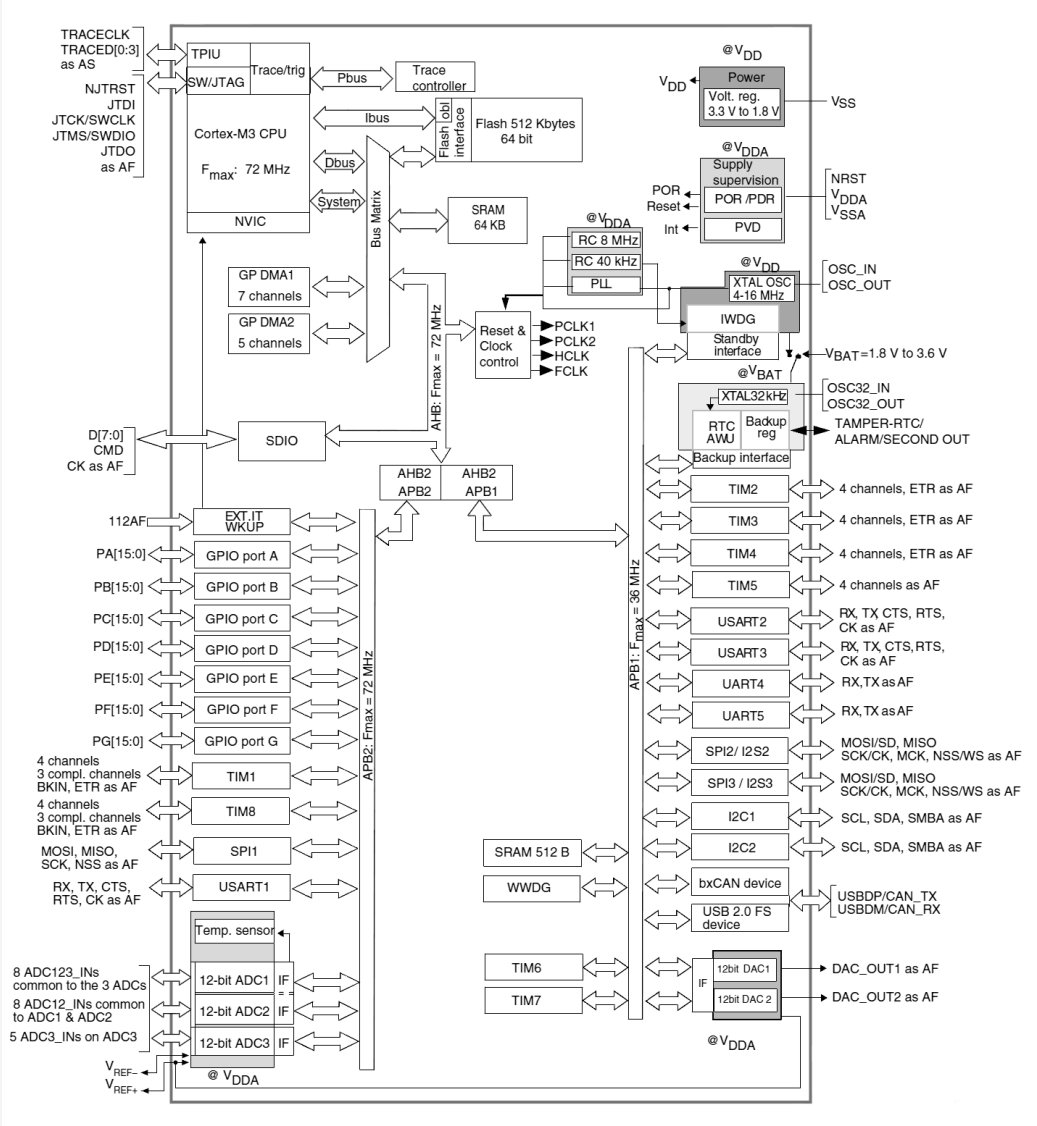
Processeur STM32F103CB

Coeur du processeur est un **ARM Cortex-M3** cadencé à 72MHz



Dimension max 10mm x 10mm

environ 1,3€pièce



Architecture des processeurs ARM

Introduction

- développé par l'équipe de R. Wilson et S. Furbe de Acorn Computer (Cambridge/Angleterre)
- Processeur inspiré par les travaux effectuées aux universités de Berkeley et Stanford ;
- 1985 : 1^{er} prototype ARM
- 1990 création de l'entreprise ARM
- ARM = Acorn Risk Machine et ensuite Advanced Risk Machine

The History of the ARM Architecture, Markus Levy, Convergence Promotions.

Architecture des processeurs ARM

Modèle économique de ARM

- Développe des architectures de micro-processeurs et des jeux d'instructions ;
- ARM ne construit aucun micro-processeur ! La technologie est licenciée à d'autres qui la mette en œuvre.

Processeurs ARM

- Supportent 32 et 64 bits ;
- L'architecture est la utilisée dans le monde :
  - 10 milliards produits en 2013 ;
  - 98% des téléphones portables contiennent au moins 1 processeur ARM

Architecture des processeurs ARM

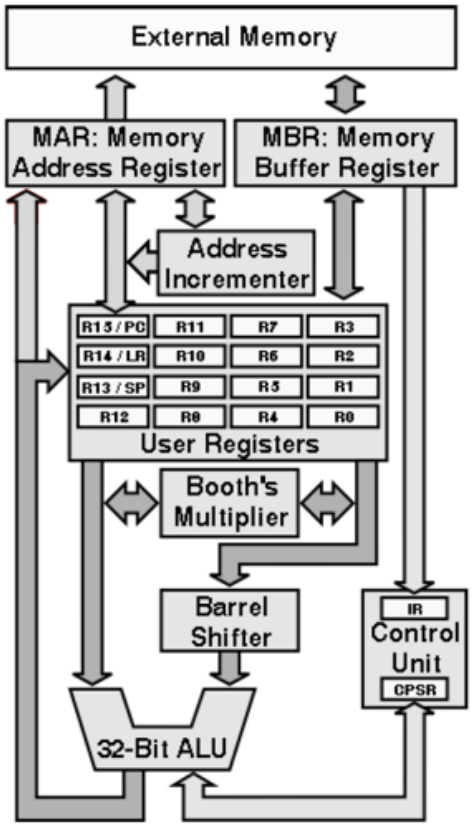
Les processeurs ARM sont partout

- ARM7TDMI(-S): Nintendo DS, Lego NXT ;
- ARM946E-S: Canon 5D Mark ii ;
- ARM1176JZ(F)-S: Raspberry Pi ;
- Cortex-A9: Apple iPhone 4S, iPad2 ;
- Cortex-A15: Nexus 10.

Architecture des processeurs ARM

Architecture ARM

- Architecture RISC pour laquelle tout passe par des registres
- Registres
  - 16 registres généraux
  - PC = 32 bits
  - CPSR (statut) = 32 bits
- Instructions de 32 bits
- Accès à 2^{32} octets (4GB) de mémoire
- “Little ou Big endian” au choix
- Opère sur 8, 16 et 32 bits



Programmation OpenCM9.04

Introduction

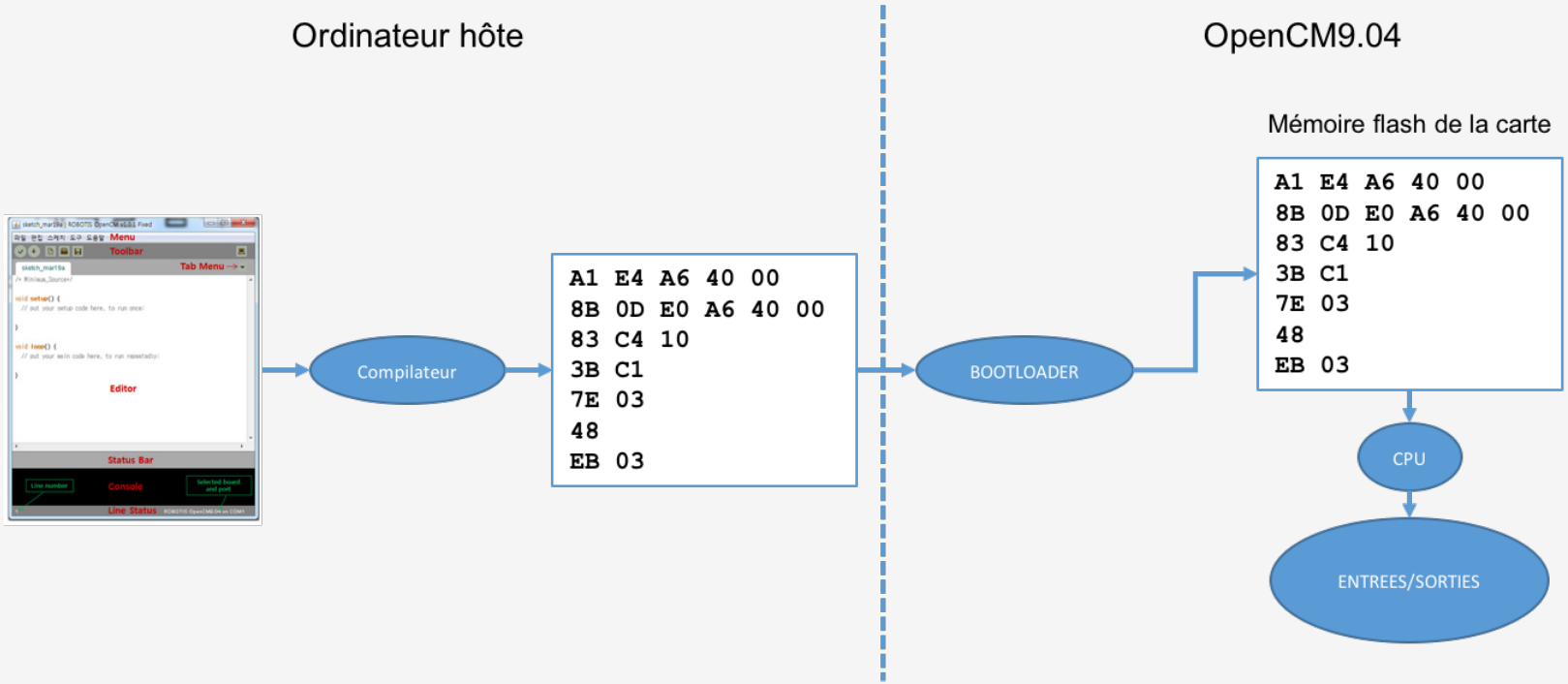
- Il existe de nombreuses méthodes pour programmer la carte OpenCM9.04 : Matlab, LabView, C/C++ ;
- Utilisation de l'environnement graphique OpenCM ROBOTIS
- OpenCM Robotis : Arduino like Integrated Development Environment
- Environnement de développement tout intégré qui permet à tous de programmer un microcontrôleur

Programmation OpenCM9.04

OpenCM Robotis

- Interface pour écrire des *croquis*
- Editeur de texte
- Compilateur intégré
- Interface de téléchargement via USB
- Interface de communication série
- Langage de programmation : proche du C/C++

Programmation OpenCM9.04



Programmation C

Introduction - Langage C

- 1972 : D. Richie et K. Thompson (Bell Labs) conçoivent le langage C pour développer UNIX ;
- 1978 : B. Kernighan et D. Richie publient la définition du C dans *The C Programming language* ;
- 1989 : Définition de la norme ANSI C et 1990 : norme ISO.

Le C est un langage *compilé* : un programme écrit en C dans un fichier texte, *fichier source*, est traduit en *langage machine* grâce à l'étape de *compilation*. Cette dernière action est réalisée par un programme appelé *compilateur*.

Programmation C

Structure d'un programme

- Un programme est une suite d'instruction terminée par ;
- **main()** est la fonction principale obligatoire
- Les variables sont déclarées et initialisées avant d'être utilisées
- Les fonctions sont des regroupements d'instructions

```
#include <myLib.h>
int main() {
  /* Declaration des variables */
  int a, b, res;
  /*Appel d'une fonction*/
  res = addition(a,b);
  /* retourne une valeur*/
  return res;  }
```

Programmation C

Les variables

En langage C, une variables est :

- un objet repéré par son nom, pouvant contenir des données, qui pourront être modifiées lors de l'exécution du programme ;
- **typée**, dont le type va définir l'espace mémoire occupé par celle-ci.

Le programmeur peut choisir le nom qu'il souhaite donner à une variable en respectant quelques règles :

- 32 premiers caractères ;
- toujours commencer par une lettre, peut contenir des caractères alpha numérique mais pas d'espace ;
- ne pas prendre des mots clés : auto, break, if, default, continue, switch, etc.

Programmation C

Déclaration d'une variable

Pour utiliser une variable il est impératif de la déclarer, c'est à dire lui donner un nom et un type :

Type `Nom_de_la_variable`;

La variable est alors déclarée, l'espace mémoire *réservé*, mais aucune valeur ne lui est affectée. Pour initialiser la variable on réalise une affectation de valeur à un espace mémoire donné :

Type `Nom_de_la_variable = donnee` ;

Programmation C

Visibilité d'une variable

- Le domaine de visibilité d'une variable est limité au bloc (d'instruction) où cette variable est déclarée, et après sa déclaration. Un **bloc d'instruction** est une suite d'instruction et repéré par des accolades ;
- Une variable déclarée dans un bloc est appelée **variable locale** ;
- Si une variable est déclarée en dehors de tout bloc (au niveau du programme principal) on dit que c'est une **variable globale** ;
- Il est interdit de déclarer deux variables avec le même nom dans un même bloc.

Programmation C

Type de données

- Chaque variable est obligatoirement typée
- Il existe un nombre limités de types :
  - entiers signés et non signés : `int, long, unsigned int, unsigned long`
  - réels : `double, float`
  - caractère : `char`
  - vide : `void`
- Possibilité de créer des tableaux, exemple :
  - `int tab[10];` est un tableau contenant 10 entiers signés, numéroté de 0 à 9

Programmation C

Expressions

- les expressions sont une combinaison de variables, d'opérateurs, de parenthèses et d'appels de fonctions ;
- opérateurs arithmétiques : $+ \ - \ * \ / \ \%$;
- opérateurs d'affectation : $=$ (égal), $+=$ (affectation avec opération) par exemple $i+=3$ équivaut à $i = i + 3$
- opérateurs d'incrémentation et de décrémentation : $++$, $--$
- opérateurs logiques :
  - $\&\&$ (ET) $||$ (OU) $!$ (NON), etc.
  - comparaison : $==$, $>$, $<$, etc.

Programmation C

Préprocesseur

Les directives du préprocesseur :

- permettent de définir des *alias* pour du texte ;
- sont traitées avant la compilation (copier-coller) ;
- permettent de créer des macros avec des arguments.

Elles commencent toujours par le caractère **#**. Exemple

- `#define PIN_INPUT_1 10`
- `#define BTN1_ON (digitalRead(PIN_INPUT_1)==HIGH)`
- `#define max(a,b) ((a)>(b)?(a):(b)`

Programmation C

Les instructions conditionnelles sont à la base des prises de décisions simples ;

Instruction conditionnelle IF

```
if (<expression>){
        <instructions1>;}
else{
        <instruction2>;}
```

Mécanisme :
l'`expression` est évaluée.

- si `expression != 0` alors les `instructions1` sont effectuées,
- sinon les `instructions2` sont effectuées,

L'instruction `else` est facultative.

Programmation C

Instruction conditionnelle : SWITCH ... CASE

```
switch(<expression>){
    case <valeur1>:
        <instructions1>;
        break;
    case <valeur2>:
        <instructions2>;
        break;
    default:
        <instructionsOther>;
        break;
}
```

Mécanisme :

l'`expression` est évaluée.

- si `expression == valeur1` alors les `instructions1` sont effectuées,
- si `expression == valeur2` alors les `instructions2` sont effectuées,
- autrement les `instructionsOther` sont effectuées,

Programmation C

Les boucles : WHILE

```
while (<expression>){
        <instructions1>;
}
```

Mécanisme :
l'expression est évaluée.

- Tant que expression est vraie les instructions1 sont exécutées

Programmation C

Les boucles : DO ... WHILE

```
do{     <instructions1>;
}while (<expression>);
```

Mécanisme :

les `instructions1` sont exécutées puis l'`expression` est évaluée.

- Tant que `expression` est vraie les `instructions1` sont exécutées

Programmation C

Les boucles : FOR

```
for(<expression1>; <expression2>; <expression3>)
 {
        <instructions1>;
}
```

Mécanisme :

Initialisation de la boucle en évaluant l'`expression1`.

- Tant que `expression2` est vraie alors exécuter les `instructions1` et évaluer l'`expression3`.

Programmation C

La notion de fonction

- Lorsqu'un programme exécute une séquence plusieurs fois ;
- Lorsque des programmes exécutent la même séquence

Alors on utilise des sous-programmes $\Rightarrow$ des **fonctions**

Défintion [Cuvillon]

- Une fonction C est définie par :
  - un nom et liste d'arguments formels (nombre, types et noms)
  - opérations à réaliser (écrites en C)
  - résultat rendu
- Une fonction est utilisée avec une **instruction d'appel** (nom de la fonction) et une liste d'**arguments**

Programmation C

Exemple de fonction

```
double  moyenne(double x, double y) {
        double mean = 0;
        mean = (x+y)/2.0;
        return mean;
  }
```

Programmation C

Utilisation d'une bibliothèque de fonction

- Collection de fonctions spécifiques dédiées à une tâche, un objet, etc. ;
- Le programmeur a accès aux fonctions de cette bibliothèque grâce aux prototypes déclarés dans le fichier d'entête ;
- Par exemple, le fichier d'entête *stdio.h* permet d'avoir accès aux fonctions d'entrées/sorties standard, *printf* permet d'afficher une valeur sur l'écran.

```c
#include <stdio.h>
int main(void)
{
        int value = 10;
        printf("%d\n", value);
        return 0;
}
```

Programmation C

Les bonnes pratiques

- Programmation structurée : il faut autant que possible découper le programme en modules
- Bien définir les variables (ne pas hésiter à mettre des noms explicites)
- Bien commenter le programme
- Faire attention à l'indentation

Programmation C++

Introduction

- Langage inventé par Bjarne Stroustrup en 1983 ;
- C'est une évolution *orienté objet* du langage C ;
- La référence : *The C++ programming Language* par Stroustrup ;
- Le C++ est généralement considéré comme un surensemble du C (mécanisme d'écriture et génération identique) ;

Extension par rapport au langage C

- les concepts orientés objet (héritage, encapsulation) ;
- surcharge de fonction, d'opérateurs ;
- templates de classes et de fonctions ;
- les références ;

Programmation C++

Exemple fichier source

```
#include <iostream>
using namespace std;
int main()
{
        cout << "BONJOUR" << endl;
        return 0;
}
```

Les fichiers sources en C++ utilisent l'extension *.cpp*,*.cc* ou *.cxx*

Programmation C++

Programmation Orientée Objet

- Contrairement à la programmation procédurale (concepts de base du langage C) qui est basée sur la notion de fonction, la programmation orientée objet va plus loin en proposant une entité unique qui est un **objet**.
- Un **objet** est avant tout une structure de données qui permet de gérer des données, de les classer et de les stocker, il regroupe également des moyens de traitement de ces données ;
- Un **objet** regroupe deux éléments de la programmation procédurale :
  - les champs : c'est l'équivalent des variables pour un programme. Ils sont définis par un type et un nom ;
  - les méthodes : ce sont des éléments qui servent d'interface entre l'objet et le programme. Généralement se sont des fonctions qui permettent de traiter les données.

Programmation C++

La notion de Classe

- Une Classe est la notion de base de la programmation orientée objet en C++;
- Un objet est un élément d'une certaine classe : on parle de l'**instance d'une classe**

L'encapsulation

- Il s'agit d'un mécanisme qui interdit d'accéder à certaines données depuis l'extérieur de la classe ;
- Une classe peut ainsi apparaître comme une boîte noire qui offre des accès aux membres que par des méthodes publiques ;
- L'encapsulation permet de déclarer des membres d'une classe *public*, *private* et *protected*

Programmation C++

Exemple de classe C++

```
#include <iostream>
using namespace std;
class CTest
{
    public :
        int getValue() { return this->value; }
        void setValue(int value) { this->value = value;}
        void print();
    private :
        int value;
};
void CTest::print()
{
    cout << "Value=" << this->value << endl;
}
int main()
{
    CTest var;
        var.setValue(10);
        var.print();
        return (0);
}
```

Programmation C++

Constructeur

Le constructeur d'un objet va se charger de mettre en place les données, associer les méthodes avec les champs et de créer le diagramme d'héritage de l'objet. Un constructeur n'est pas nécessaire pour un objet statique, et un objet peut avoir plusieurs constructeurs.

Destructeur

Le destructeur est le pendant du constructeur. Il va se charger de détruire l'instance de l'objet. Un objet statique n'a pas nécessairement de destructeur, un objet peut avoir plusieurs destructeurs.

Programmation C++

Exemple de classe C++

```
class chaine{
    char * s;
public:
        chaine(void);           // Le constructeur par defaut.
        chaine(unsigned int);   // Le constructeur avec une valeur par defaut
        ~chaine(void);          // Le destructeur.
};

chaine::chaine(void){
        s=NULL;                 // Initialisation de la chaine  NULL
}
chaine::chaine(unsigned int Taille){
        s = new char[Taille+1];     // Allocation
        s[0]='\0';                      // Initialisation
}
chaine::~chaine(void){
        if (s!=NULL)
            delete[] s;
}
chaine s1;      // une chaine de caracteres
                        // non initialisee.
chaine s2(200);     // Instancie une chaine de caracteres
                        // de 200 caracteres.
```

Eléments de programmation OpenCM9.04

Introduction à l'environnement de développement

Le code est organisé en **sketch**es (croquis). Chaque sketch est représenté par un répertoire et au moins un fichier *.ino* ;

-  Lance la compilation du sketch et affiche les erreurs si il y en a ;
-  Lance la compilation du sketch, et transfert l'exécutable sur le microcontroleur si il n'y a pas d'erreur;
-  Création d'un nouveau sketch ;
-  Ouverture d'un sketch existant ;
-  Enregistre le sketch courant ;
-  Ouverture d'une fenêtre qui donne accès au moniteur série, de cette manière l'utilisateur peut interagir avec la carte

Eléments de programmation OpenCM9.04

Principes de programmation

- la fonction *main()* est déjà implémentée ;
- le programmeur doit écrire les fonctions :
  1. déclaration et initialisation des variables ;
  2. *setup()* : fonction appelée une seule fois, qui permet d'initialiser les ports et le protocole de communication ;
  3. *loop()* : fonction appelée de manière périodique. Attention, les variables déclarées dans cette fonction seront écrasées et recrées chaque fois que la fonction est appelées ;
- le langage de programmation se rapproche du C et C++ (utilisation de classes, héritage, etc.)

Eléments de programmation OpenCM9.04

Principes de programmation

```
__attribute__(( constructor )) void premain() {
    init();
}
int main(void) {
    setup();
    while (1) {
        loop();
    }
    return 0;
}
```

fichier *main.cpp* dans core de ROBOTIS.

Eléments de programmation OpenCM9.04

Type de données spécifiques

- 1 octet : uint8_t, int8_t (byte, char)
- 2 octets : int16_t, uint16_t (int)
- 4 octets : int32_t, uint32_t (long)

Techniques de programmation

- Taille mémoire limitée : dimensionnement des variables
- Utilisation des constantes (mots clés *const* et *define*) pour libérer de la mémoire d'exécution
- déclarer des variables globales

Eléments de programmation OpenCM9.04

Gestion des entrées/sorties

- La carte OpenCM9.04 permet d'interagir avec le monde réel
- avec la fonction *setup()* il est possible de configurer les entrées/sorties de la carte
- exemple :

```
void setup() {
    // Set up the built-in LED pin as an output:
    pinMode(BOARD_LED_PIN, OUTPUT);
}
void loop() {
    digitalWrite(BOARD_LED_PIN, HIGH); // set to as HIGH LED is turn-off
    delay(100);          // Wait for 0.1 second
    digitalWrite(BOARD_LED_PIN, LOW);  // set to as LOW LED is turn-on
    delay(100);          // Wait for 0.1 second
}
```

Eléments de programmation OpenCM9.04

Gestion des Entrées/Sorties : Objet Dynamixel

- La classe Dynamixel offre des méthodes qui permettent d'interagir avec les moteurs ;
- Voici quelques méthodes *public* : **Accès bas niveau**

```
byte readByte(byte bID, word bAddress);
byte writeByte(byte bID, word bAddress, byte bData);
word readWord(byte bID, word bAddress);
byte writeWord(byte bID, word bAddress, word wData);

byte writeDword( byte bID, word wAddress, uint32 value );
uint32 readDword( byte bID, word wAddress );
```

Eléments de programmation OpenCM9.04

Gestion des Entrées/Sorties : Objet Dynamixel

- La classe Dynamixel offre des méthodes qui permettent d'interagir avec les moteurs ;
- Voici quelques méthodes *public* : **Configuration des moteurs**

```
void setID(byte current_ID, byte new_ID);
void setBaud(byte bID, byte baud_num);
void controlMode(byte bID, byte mode); // change wheel, joint
byte controlMode(byte bID); // return current mode
void jointMode(byte bID);
void wheelMode(byte bID);
void maxTorque(byte bID, word value);
word maxTorque(byte bID);
```

Eléments de programmation OpenCM9.04

Gestion des Entrées/Sorties : Objet Dynamixel

- La classe Dynamixel offre des méthodes qui permettent d'interagir avec les moteurs ;
- Voici quelques méthodes *public* : **Dialogue avec les moteurs**

```
byte setPosition(byte ServoID, int Position, int Speed);
void goalPosition(byte bID, int position);
void goalSpeed(byte bID, int speed);
void goalTorque(byte bID, int torque);
int getPosition(byte bID);
int getSpeed(byte bID);
byte isMoving(byte bID);
```

Exemple de croquis - DYNAMIXEL

```
/* Configuration liaison serie pour la communcation avec DXL */
#define DXL_BUS_SERIAL1 1  //Dynamixel sur Serial1(USART1)  <-OpenCM9.04
#define DXL_BUS_SERIAL2 2  //Dynamixel sur Serial2(USART2)  <-LN101,BT210
#define DXL_BUS_SERIAL3 3  //Dynamixel sur Serial3(USART3)  <-OpenCM 485EXP
/* Definition identifiant Dynamixel */
#define ID_NUM 1
/* Identifiant du registre contenant la consigne de Position*/
#define GOAL_POSITION 30 // Definit dans la datasheet

Dynamixel Dxl(DXL_BUS_SERIAL1);

void setup() {
    // Dynamixel 2.0 Baudrate -> 0: 9600, 1: 57600, 2: 115200, 3: 1Mbps
    Dxl.begin(3);
    Dxl.jointMode(ID_NUM); //jointMode() est pour l'utilisation en mode position
}
void loop() {
    //dynamixel avec ID 1 en position 0
    Dxl.writeWord(ID_NUM, GOAL_POSITION, 0);
    // Attente  1 seconde (1000 millisecondes)
    delay(1000);
    // dynamixel avec ID 1 en position 300
    Dxl.writeWord(ID_NUM, GOAL_POSITION, 300);
    // Attente  1 seconde (1000 millisecondes)
    delay(1000);
}
```

Eléments de programmation OpenCM9.04

Gestion de la périodicité

La périodicité d'appel de la fonction `loop()` va dépendre du matérie et des instructions qui seront exécutées dans cette fonction. Pour cadencer cette boucle on peut :

- placer une temporisation bloquante `delay(time)` time en millisecondes dans la fonction `loop()` ;
- utiliser la fonction `millis()` qui permet d'obtenir le mesure de temps courante ;
- utiliser des interruptions logicielles associées à un timer ;

Eléments de programmation OpenCM9.04

Gestion de la périodicité - `delay()`

```
void setup() {
    /*instructions d'initalisation*/
}
void loop() {
    /* code principal qui sera repete en boucle*/
    delay(100); /* on cadence la boucle : 100 ms +/- temps pour les instructions du code principal*/
}
```

Eléments de programmation OpenCM9.04

Gestion de la périodicité - millis()

```
unsigned long interval=100;
unsigned long previousMillis=0;
void setup() {
    /*instructions d'initalisation*/
}
void loop() {
    unsigned long currentMillis = millis();
    if((unsigned long)(currentMillis - previousMillis) > interval) {
        /* code principal qui sera repete en boucle*/
        /* sauvegarde instant precedent*/
        previousMillis = currentMillis;
    }
}
```

Eléments de programmation OpenCM9.04

Gestion de la périodicité - interrupts()

```
#define LOOP_RATE 1000000    // in microseconds
HardwareTimer Timer(1); // Instance de la classe HardwareTimer sur le  timer device 1
void setup() {
    // Pause du timer pendant la configuration
    Timer.pause();
    // Definir la periode
    Timer.setPeriod(LOOP_RATE); // en microsecondes
    // Activation de l'interruption sur le canal 1
    Timer.setMode(TIMER_CH1, TIMER_OUTPUT_COMPARE);
    Timer.setCompare(TIMER_CH1, 1);  // Compte Interruption 1 apres mise a jour
    Timer.attachInterrupt(TIMER_CH1, handler_loop);
    // RAZ
    Timer.refresh();
    // Demarrage du timer
    Timer.resume();
}
void loop() {
}
void handler_loop(void) {
        /* code principal qui sera repete en boucle*/
}
```

Attention : les interruptions sont des mécanismes difficiles a gérer car elles n'évitent pas la réentrance.

Eléments de programmation OpenCM9.04

Pour aller plus loin ...

Dans les bibliothèques de fonctions, il y a `MapleFreeRTOS.h` qui permet de faire du multi-tâches. Cette bibliothèque s'appuie sur FreeRTOS qui est un système d'exploitation temps-réel open-source et gratuit.

```
BaseType_t xTaskCreate(     TaskFunction_t pvTaskCode,  //fonction associee a la tache
                        const char * const pcName,  // nom de la tache
                        unsigned short usStackDepth,  // allocation memoire
                        void *pvParameters,  //parametres
                        BaseType_t uxPriority,  // priorite
                        TaskHandle_t *pxCreatedTask);    // utilisee pour transmettre un appel  a la tache
void vTaskStartScheduler( void ); // demarrer les taches
void vTaskEndScheduler( void ); // arreter les taches
```

Eléments de programmation OpenCM9.04

```
#include <MapleFreeRTOS.h>
void setup(){
    xTaskCreate( vLedTask, ( signed char * ) "Led", configMINIMAL_STACK_SIZE, NULL, 1, NULL );
    xTaskCreate( vUartTask, ( signed char * ) "Uart", configMINIMAL_STACK_SIZE, NULL, 1, NULL );
    //start run tasks
    vTaskStartScheduler();
}
void loop(){
}
void vLedTask( void *pvParameters )
{
    for( ;; ){
            toggleLED();
            delay(100);
    }
}
void vUartTask( void *pvParameters )
{
    SerialUSB.begin();
    for( ;; )
    {
            SerialUSB.println("Hello World");
            delay(100);
    }
}
```

Projet

Objectif

Contrôler le système robotique 5 barres pour suivre des trajectoires et faire des dessins en utilisant OpenCM9.04 et myRIO.

Projet

Déroulement

Nous proposons de travailler en mode projet : c'est à dire en autonomie avec des séances où un enseignant sera présent. Pour le déroulement du projet nous imposons des tâche qu'il conviendra de réaliser :

- Tâche 1 : Modélisation et simulation du système robotique (M. Barbé);
- Tâche 2 : Simulation du contrôle du système robotique (M. Barbé);
- Tâche 3 : Implémentation du contrôle avec OpenCM9.04 (M. Barbé);
- Tâche 4 : Implémentation du contrôle avec myRIO (M. Roth);
- Evaluation du travail avec présentation avec support ;

Projet

Tâche 1 : Modélisation et simulation du mécanisme

- établir les modèles géométriques direct et inverse du mécanisme 5 barres en utilisant le formalisme proposé ;
- établir le modèle cinématique direct du mécanisme ;
- Déterminer les singularités du mécanisme ;
- Définir une zone de travail adéquate ;
- Attention le positionnement du crayon sur le mécanisme est légèrement décalé par rapport à l'organe terminal

Certaines de ces tâches peuvent être menées en parallèle, donc il y a du travail pour tous !

Projet

Tâche 2 : Simulation du mécanisme

- Utiliser l'outil de votre choix (Matlab ou mapple ou autre)
- Implémenter les modèles du mécanisme 5 barres ;
- Deux scénarios :
  - Déplacement du robot du point courant vers un nouveau point dans l'espace (minimum de jerk) ;
  - Déplacement du robot selon une courbe paramétrée en temps, par exemple :
    - un cœur : $x(t) = sin^3(t)$ et $y(t) = cos(t) - cos^4(t)$ (normalisée)
    - cercle unité : $x(t) = cos(t)$ et $y(t) = sin(t)$

Projet

Tâche 3 : Implémentation du contrôle avec OpenCM9.04

- Découverte de l'environnement de développement OpenCM ;
- Prise en main de l'ensemble carte OpenCM9.04 et moteur Dynamixel ;
- Implementer les différents modèles ;
- Mettre en oeuvre les deux scénarios simulés ;
- ATTENTION les moteurs sont couplés à une structure mécanique ... qui casse !

Projet

Tâche 4 a: Implémentation du contrôle avec myRIO

- Créer un programme pour le myRIO permettant de piloter les moteurs en créant une IHM qui permet :
  - contrôler la position (deg) et la vitesse (rpm) de chaque moteur ;
  - tracer une courbe paramétrée, par exemple un coeur et un cercle ;

Contraintes et cahier des charges

- Utiliser le projet de départ fourni et la bibliothèque Dynamixel fourni ;
- La structure du programme est libre mais le VI principal doit se nommer RTmain.vi à la racine du projet ;
- Tracer la trajectoire de A_3 : indicateur GraphXY ou Plot2D ;
- Gestion d'une erreur sur le flux de données : arrêt du programme ;
- Bouton d'arrêt d'urgence (stop le programme proprement).

Projet

Tâche 4 b: Implémentation du contrôle avec myRIO

- Créer un programme pour le myRIO permettant de piloter les moteurs à partir de points dans un tableau 2D ;
- Le mouvement de la souris doit être captée en local et envoyée au myRIO (possibilité d'utiliser les variables réseau) ;

Contraintes et cahier des charges

- Utiliser le projet de départ fourni et la bibliothèque Dynamixel fourni ;
- La structure du programme est libre mais le VI principal doit se nommer RTmain.vi à la racine du projet ;
- Tracer la trajectoire de A_3 : indicateur GraphXY ou Plot2D ;
- Gestion d'une erreur sur le flux de données : arrêt du programme ;
- Bouton d'arrêt d'urgence (stop le programme proprement).

Projet AIDE

1/4

La génération de trajectoire en robotique consiste à déterminer la trajectoire permettant de relier deux points dans l'espace (cartésien ou articulaire) en respectant les contraintes et les critères fixés.
Hypothèse :

- Déplacement du point $A = (x_A, y_A)$ vers le point $B = (x_B, y_B)$
- les vitesses et les accélérations aux points A et B sont nulles.
- les déplacements sont réalisés en un temps limité, que nous fixerons à t_f.

Méthode basée sur le minimum de jerk dans l'**espace articulaire**. On cherche à minimiser la fonction de coût :

$$C = \frac{1}{2} \int_0^{t_f} \dddot{q}(u) du$$

On pose

$$q(\tau) = a\tau^5 + b\tau^4 + c\tau^3 + d\tau^2 + e\tau + f \text{ avec}$$

polynôme d'ordre 5 normalisé avec $\Delta q = q(1) - q(0)$ et $\tau = \frac{t}{t_f}$

Projet AIDE

2/4

Sachant que :

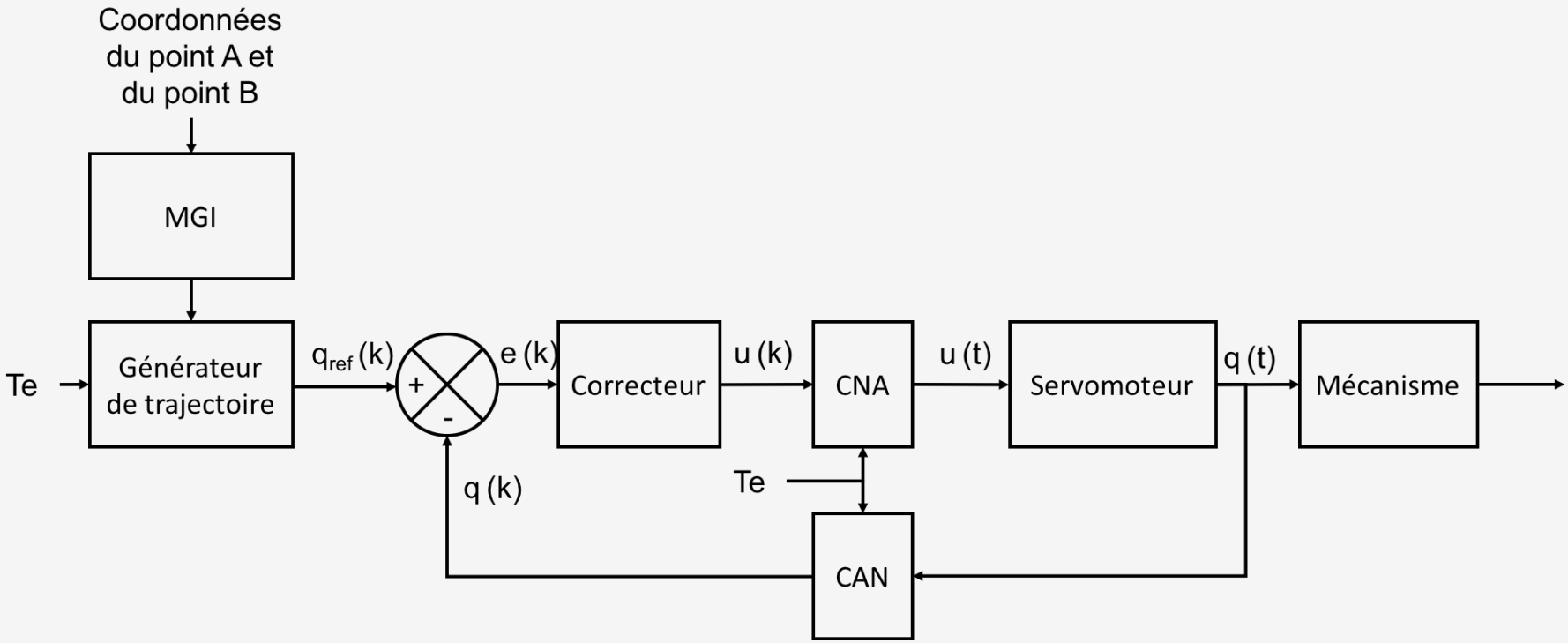
$$\begin{aligned} \dot{q}(0) &= 0 \\ \dot{q}(1) &= 0 \\ \ddot{q}(0) &= 0 \\ \ddot{q}(1) &= 0 \end{aligned}$$

Le polynôme décrivant la trajectoire peut alors s'écrire :

$$q(\tau) = q(0) + \Delta q \left(6\tau^5 - 15\tau^4 + 10\tau^3\right)$$

Projet AIDE

3/4



Projet AIDE

4/4

1: $[q_1(0)\ q_4(0)]^T \leftarrow MGI(x_A, y_A)$
2: $[q_1(t_f)\ q_4(t_f)]^T \leftarrow MGI(x_B, y_B)$
3: $k = 0$
4: $\Delta q_1 = q_1(t_f) - q_1(0)$
5: $\Delta q_4 = q_4(t_f) - q_4(0)$
6: **while** $kT_e < t_f$ **do**
7: $\quad \tau = \frac{kT_e}{t_f}$
8: $\quad q_1^{ref}(k) = q_1(0) + \Delta q_1 \left(6\tau^5 - 15\tau^4 + 10\tau^3\right)$
9: $\quad q_4^{ref}(k) = q_4(0) + \Delta q_4 \left(6\tau^5 - 15\tau^4 + 10\tau^3\right)$
10: $\quad k++$
11: $\quad$ Conversion en discret pour envoie vers moteurs
12: $\quad$ Envoyer $q_1^{ref}(k)\ q_4^{ref}(k)$ aux moteurs
13: **end while**
