Module Robotique

Initiation à la robotique mobile

Frédéric Giamarchi
IUT de Nîmes – CSU Montpellier Nîmes
Université de Montpellier

La Robotique en IUT GEII

- Association d'une carte électronique avec un environnement mécanique simple (notion de systèmes embarqués)
- Lecture de capteurs simples et intelligents (UART, I2C, SPI)
- Contrôle de moteurs simples (CC, pas à pas, servomoteur)
- Notion d'intelligence artificielle
- Participation à des compétitions de robotique

Objectifs du cours

- Appréhender les spécificités des systèmes embarqués par rapport aux domaines de l'informatique industrielle.
- Découvrir les techniques de programmation liées aux systèmes embarqués (systèmes multitâches coopératifs, machine d'états).
- Utiliser des interfaces homme machines avancées (IHM, Liaison série, Bluetooth, Wifi, Afficheurs LCD, supervision Labview).
- Tester des algorithmes de traitements (moyenne, barycentre,…).
- Aborder la notion de déverminage (Stop & Go, IHM).
- Application à une maquette robotique (Intelligence Artificielle).

Systèmes Embarqués (S.E.)

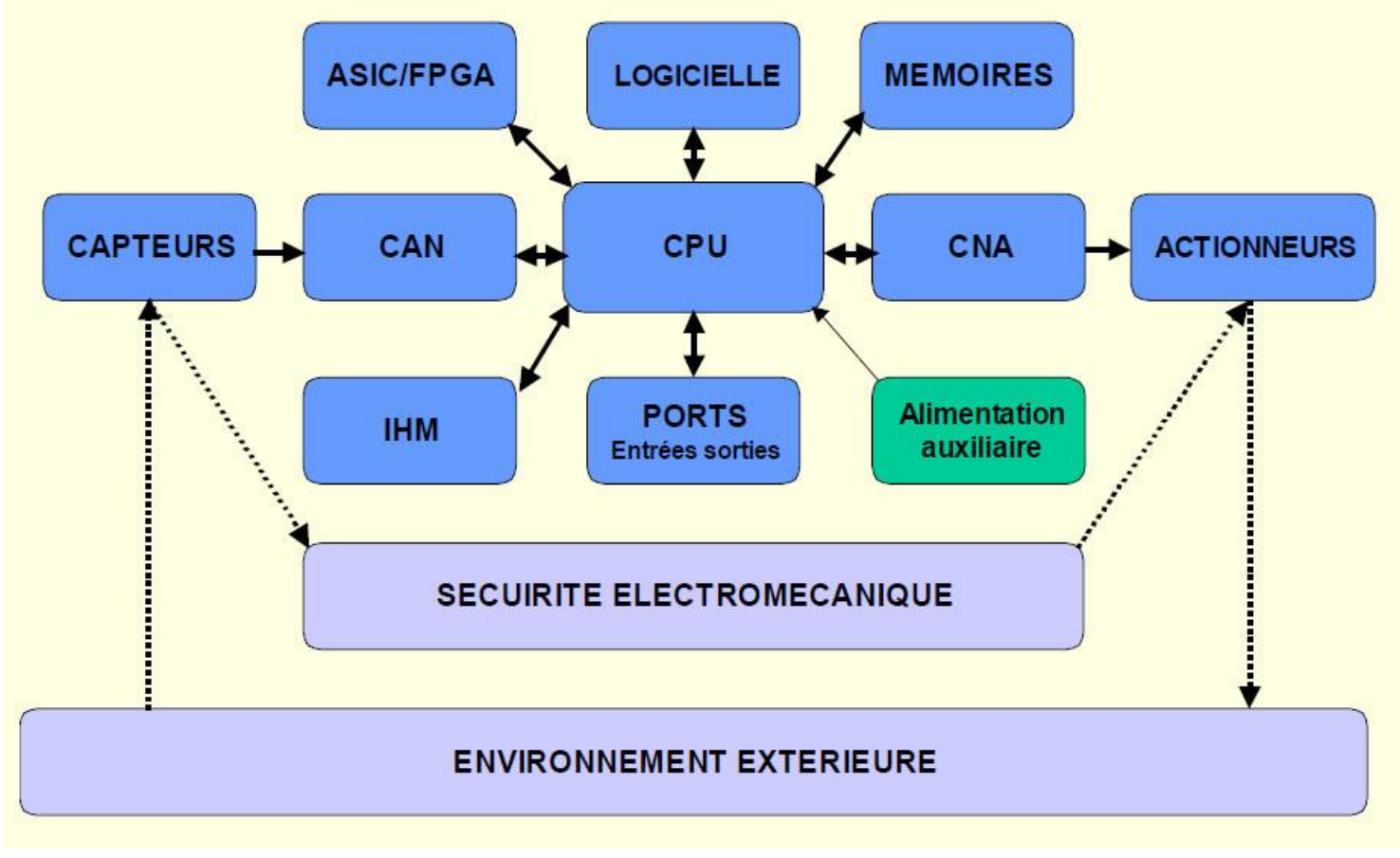
- Différents secteurs industriels
  - Télécoms (Smartphone, Internet)
  - Aéronautique (Avions, Radar, Satellite)
  - Automobile (GPS, ABS, ESP)
  - Construction électrique (Systèmes de surveillance, Gestion électrique, contrôle d'accès)
  - Transport ferroviaire
  - Cartes à puce (et RFID)
- **Explosion des besoins** : doublement du nombre d'objets intelligents et de la complexité embarquée tous les 2/3 ans
- Chaque industrie est confrontée à des **contraintes spécifiques**
- Mais les **enjeux sont similaires**
  - Maîtrise des coûts
  - Maîtrise de la complexité
  - Pérennité des solutions développées

Introduction aux systèmes embarqués

- Définition :
  - **« Embedded system»** : tout système conçu pour résoudre un problème ou une tâche spécifique, mais n'est pas un ordinateur d'usage général.
  - **Utilisent généralement un microcontrôleur**, combiné avec d'autres matériel et logiciel pour résoudre un problème de calcul spécifique.
  - C'est un système électronique et informatique **autonome** ne possédant pas d'entrées-sorties standards (clavier, écran, souris).
  - **Le système matériel et l'application sont intimement liés** et noyés dans le matériel et ne peuvent être discernables comme dans un PC.
  - On dit que le système est **enfoui** (traduction de «embedded») dans **une autre fonction**.
  - La conception de ces systèmes est **fiable** (avion, système de freinage ABS) car ils sont conçus pour des applications uniques et qu'il n'est plus possible d'intervenir sur le logiciel ensuite.

Introduction aux systèmes embarqués

- Système embarqué typique



Introduction aux systèmes embarqués

- Caractéristiques d'un système embarqué
  - Fonctionnement en **temps réel**
  - Réactivité : les opérations de calcul doivent être exécutées en réponse à un évènement extérieur (**interruption matérielle**).
  - La plupart des systèmes sont « multirate », c'est-à-dire traitement des informations à **différents rythmes**.
  - Faible encombrement, poids réduit, **consommation minimisée** (batterie)
  - **Environnement sévère** (température, vibrations, variations d'alimentation, interférences HF, corrosion, eau, feu, radiations,…).
  - **Le système n'évolue pas dans un environnement contrôlé** (évolutions des caractéristiques des composants dans le temps).
  - **Sureté** de fonctionnement (système opérationnel même lorsqu'un composant est défaillant).
  - Beaucoup de systèmes embarqués sont fabriqués en **grande série** et doivent avoir des prix de revient faibles.

Techniques de programmation

- Ne pas utiliser les fonctions de temporisations
- Utiliser des fonctions avec transfert de paramètres entre fichiers
- Utiliser des structures
- Découper une fonction longue en un diagramme d'états
- Programmation synchrone
- Machine d'états

Ne pas utiliser les fonctions de temporisations

- **Remplacer les fonctions de temporisations**.
- Pourquoi ?
  - Une temporisation ne fait rien d'autres qu'attendre !
  - Il n'est donc pas possible de faire deux actions indépendamment l'une de l'autre.
- Exemple : La temporisation de lumière d'un escalier
  - Code :

```
if  (Bouton_Poussoir == 1)          // Appui sur le bouton poussoir
    {   Del = 1;                    // La del s'éclaire
        Delay_ms (15000);           // Temporisation de 15s
        Del = 0;    }               // La del s'éteint
```

  - Si on appuie sur le bouton, la lumière s'éclaire pendant 15s, puis s'éteint. Pendant ce temps, un autre appui n'a aucun effet.
  - Il n'est donc pas possible d'allonger cette temporisation par un nouvel appui. Le système n'est pas redéclenchable.

Remplacer les fonctions de temporisations

- **Ne pas utiliser les fonctions de temporisations**.

- Comment faire autrement ?
  - En comparant une variable temps avec une constante choisie.
  - Une routine d'interruption Timer incrémente une variable temps.
  - A la place de la fonction de temporisation, on mémorise la variable temps dans une nouvelle variable « tempo ».
  - Puis on compare la variable temps avec notre nouvelle variable tempo à laquelle on ajoute la durée de la temporisation voulue.

Remplacer les fonctions de temporisations

- **Ne pas utiliser les fonctions de temporisations**.

- Code :

```
// Var_1ms sera incrémentée dans la routine d'interruption toutes les millisecondes.

int   Tempo;          //  déclaration d'une variable de type 16 bits, afin de compter jusqu'à 65535

while (1)                                   // Boucle infinie du programme
{
      if  (Bouton_Poussoir == 1)                    // Appui sur le bouton poussoir
      {
         Led = 1;                                   // La Led s'éclaire
         Tempo = Var_1ms + 15000;                   // Enregistre la valeur actuelle de la variable temps dans la
       }                                            // variable Tempo + la durée de la temporisation souhaitée

      if (Var_1ms == Tempo)                         // Test si la variable temps a atteint la tempo prévue
         Led = 0;                                   // La Led s'éteint
}
```

Utilisation de fonctions avec paramètres

- **Utiliser des fonctions avec paramètres**.
  - Une fonction permet de structurer une routine.
- Déclaration d'une fonction
  - Code d'une fonction simple :

```
void Fonction_A (void)
 {
      // Code de la fonction
}
```

  - Code d'une fonction qui attend deux paramètres de type entier:

```
void Fonction_A (int parametre_A, int parametre_B)
 {
      // Code de la fonction
}
```

  - Code d'une fonction qui retourne un paramètre de typer entier:

```
int Fonction_A (void)
 {
Int   Parametre;
      // Code de la fonction
      return (Parametre);
}
```

Utilisation de structures

- Une structure pour chaque fonction ou comportement

```
struct    Mes_Fonctions
 {
    uint8_t       etat;
    uint16_t      compteur;
    uint16_t      tempo;
    uint8_t       memoire;
    uint8_t       valeur;
}
```

```
struct    Comportement
{
    uint8_t          etat;
    uint16_t         compteur;
    uint16_t         tempo;
    uint8_t          mémoire;
    uint8_t          flag;
    signed int8_t    vit_dro;
    signed int8_t    vit_gau;
}
```

- Chaque fonction ou comportement est déclaré ensuite

```
struct    Mes_Fonctions     Cœur;
struct    Mes_Fonctions     CodeErreur;

struct    Comportement      SL;        // déclaration des variables associées au
                                       // comportement Suiveur de Ligne (SL)

SL.etat = 0;                // La variable etat associée au comportement SL est mise à 0;
SL.compteur ++;             // La variable compteur s'incrémentée de +1
```

Diagramme d'états

- **Ne pas écrire des routines trop longues.**

- Pourquoi ?
  - Une routine trop longue ne laisse pas de temps aux autres routines !
  - Il n'est donc pas possible de faire deux actions indépendamment l'une de l'autre (comme pour les fonctions de temporisations).

- Comment faire autrement ?
  - En découpant la routine (tâche), en 2 ou plus d'états ou sous-tâches.
  - Diagramme d'états (procédure à états positionnées, machine à variables d'états), sont utilisées lorsqu'une équation ou une alternative binaire ne peuvent pas être appliquée.
  - Principe d'un Grafcet avec étapes et transitions.

Diagramme d'états : générer un code d'erreur

- Exemple : x clignotements répétés
- Variables :
  - CodeErreur.etat
  - CodeErreur.compteur
  - CodeErreur.tempo
  - CodeErreur.memoire
- Etat 0 : Attente d'activation
  - Si nbre clignotement > 0 → Etat = 1 et on mémorise la valeur du compteur et le nbre de clignotement
- Etat 1 : Led allumée
  - Si Tempo On atteinte → Etat 2 et on mémorise la valeur du compteur
- Etat 2 : Led éteinte
  - Si Tempo Off atteinte → Etat = 1 et on mémorise la tempo On. On décrémente la mémoire et si = 0 → Etat = 0

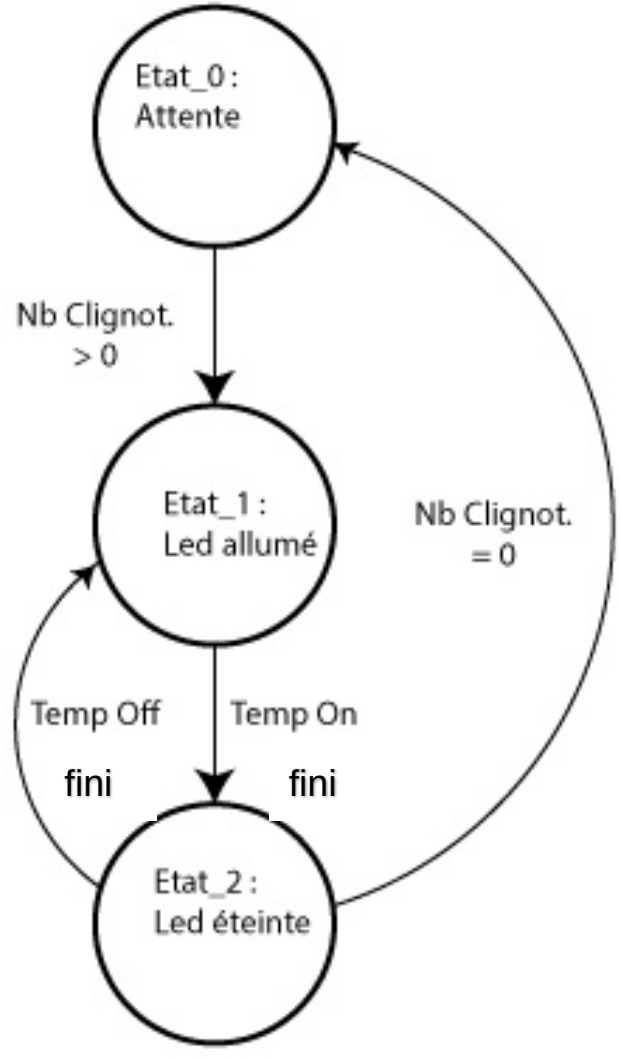
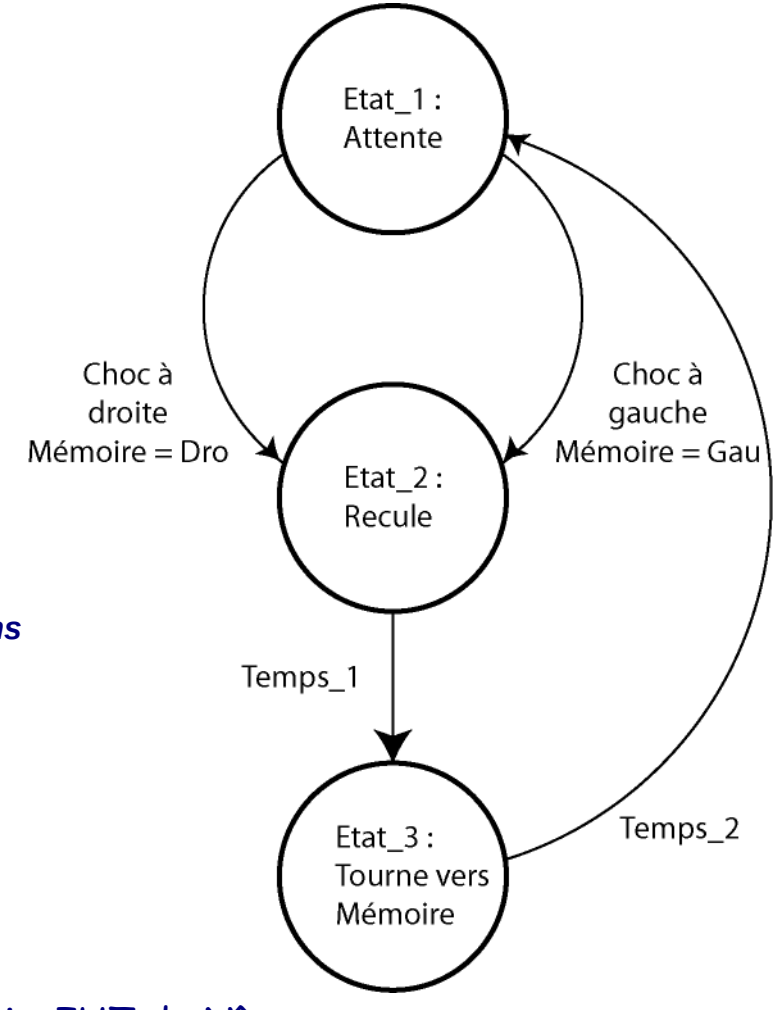


Diagramme d'états

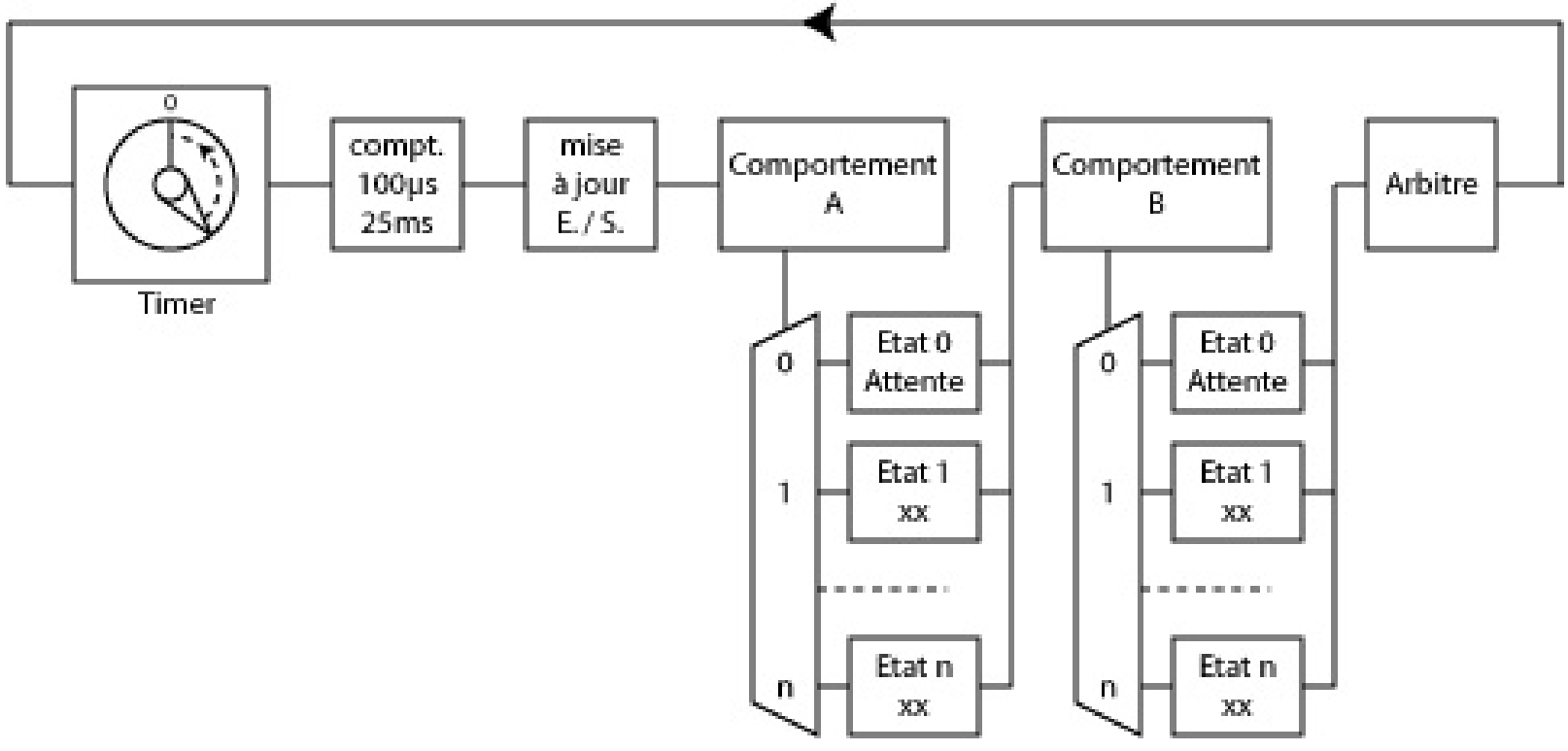
- Exemple : Action sur choc détecté à l'avant d'un robot.
- Code :

```
enum Etat {Attente, Recule, Tourne} Etat_Choc;
 void   Choc (void)
{     switch (Etat_Choc)
      {
      case Attente:     // Code pour l'état Attente
         {       if (Choc != 0)
                 {          Etat_Choc = Recule;
                            Mémoire = Choc;
                            tempo = compteur; }
                            break;  }
      case Recule:      // Code pour l'état Recule
         {       Cmd_Moteurs (-10,-10);
                  if (compteur == tempo + 50) // 50x10ms
                            Etat_Choc = Tourne;
                            break;  }
      case Tourne:      // Code pour l'état Tourne
         {
                            break;  }
      }
}
```



La programmation synchrone

- Système multitâche coopératif
- Utilisation d'une routine d'interruption pour synchroniser les tâches.
  - Timer réglé sur 1ms



Programmation performante

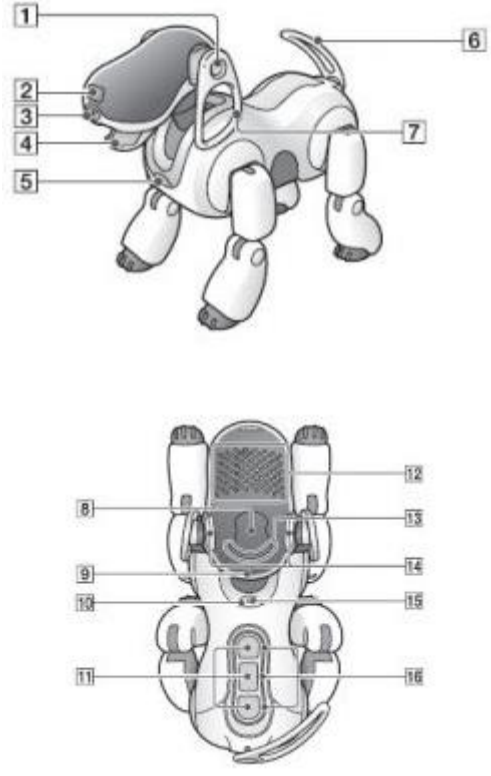
- Systèmes temps réel → multi-tâches
- Intelligence Artificielle → robuste
- Multiprocesseurs → réduction de la taille du programme
- Basse fréquence → autonomie, mode sleep

Lecture des entrées

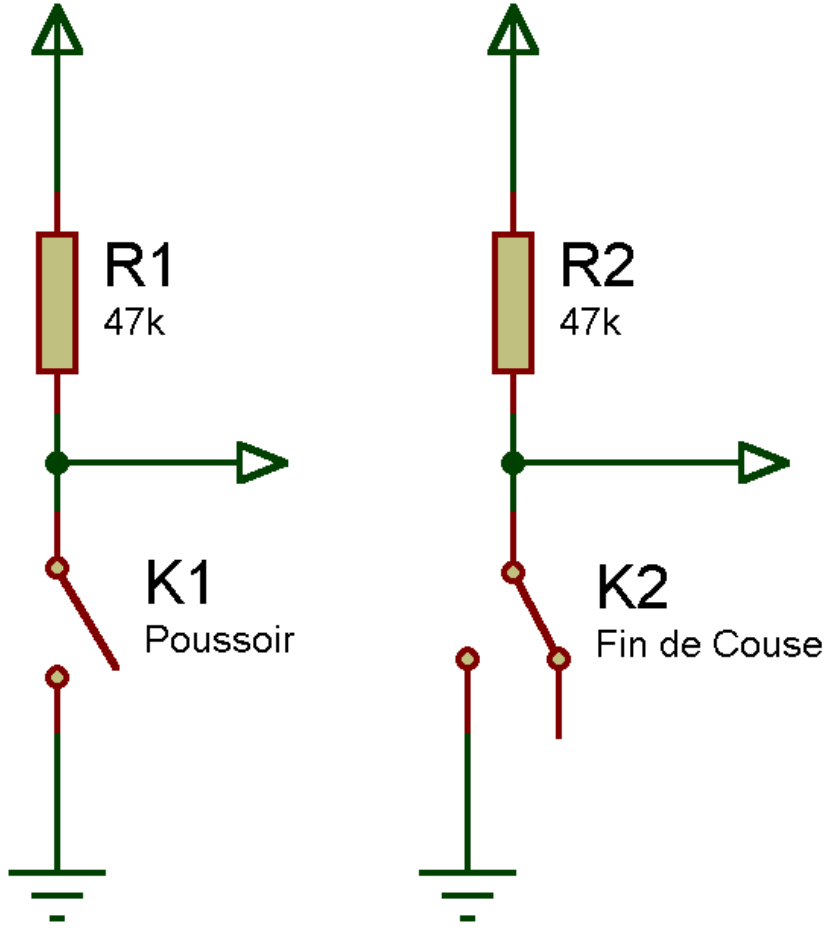
- Nature des entrées
  - Logique
  - Analogique
  - Liaison série (UART, I2C, SPI, OneWire, …)

- Capteur virtuel par fusion d'entrées
  - Deux capteurs de chocs associés permettent de détecter la position du point d'impact.
  - Plusieurs capteurs de réflexion infrarouge permettent de définir la position d'un mobile par rapport à une ligne.
  - Extraction d'informations physiques non disponibles directement à partir de capteurs élémentaires.

Lecture des entrées



Lecture des entrées

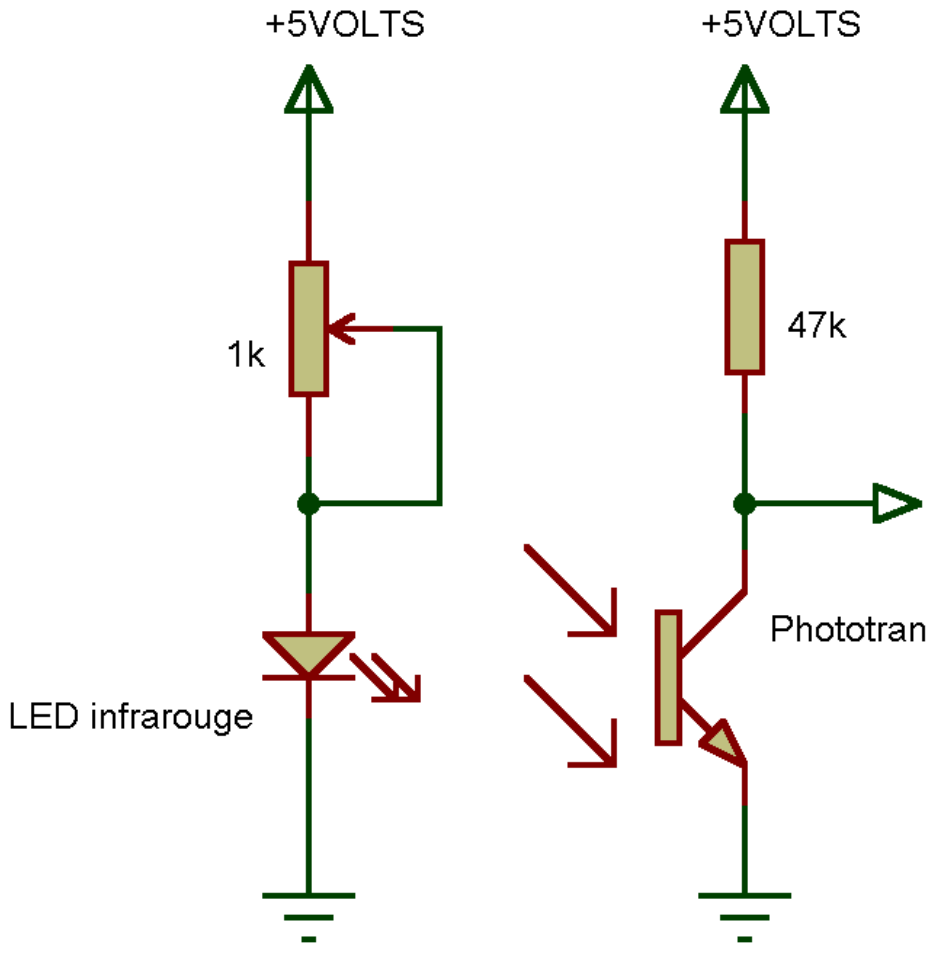


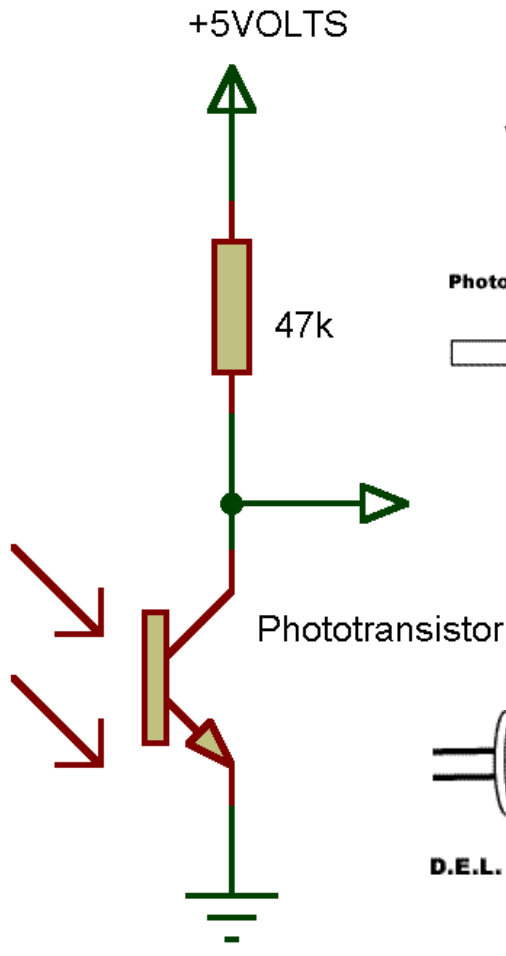
Fonctionnement

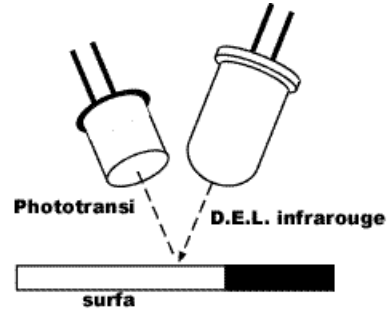
1 si ouvert

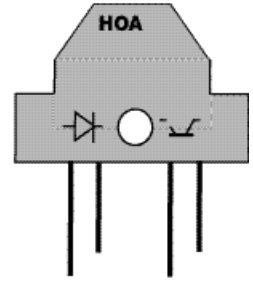
0 si fermé

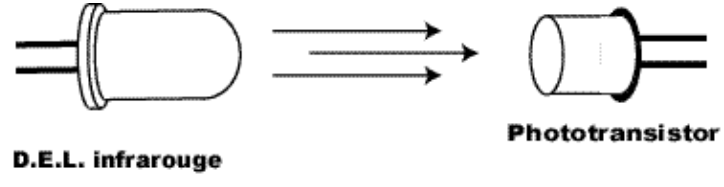
Lecture des entrées



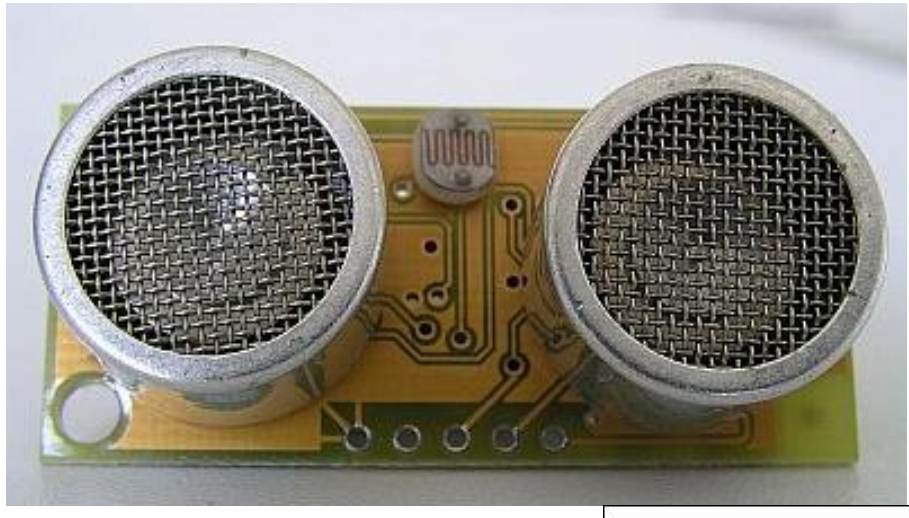




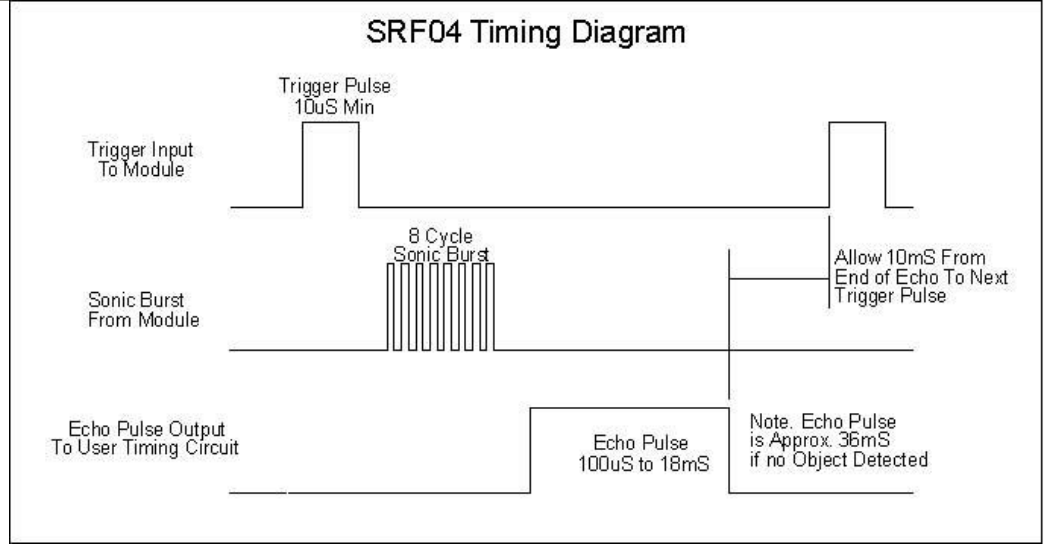




Lecture des entrées

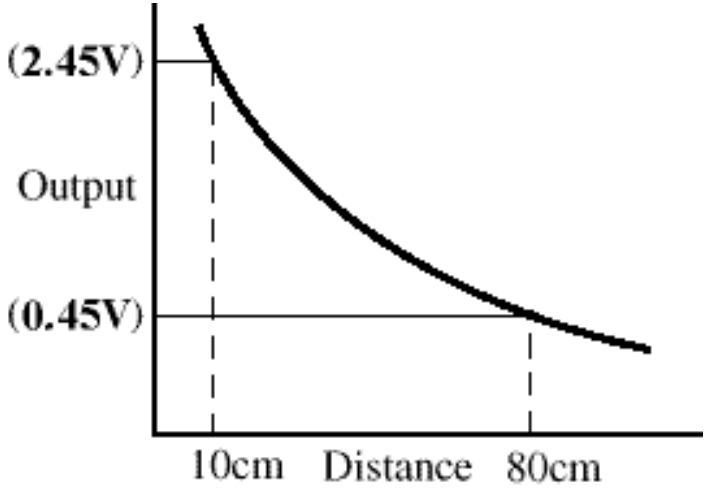


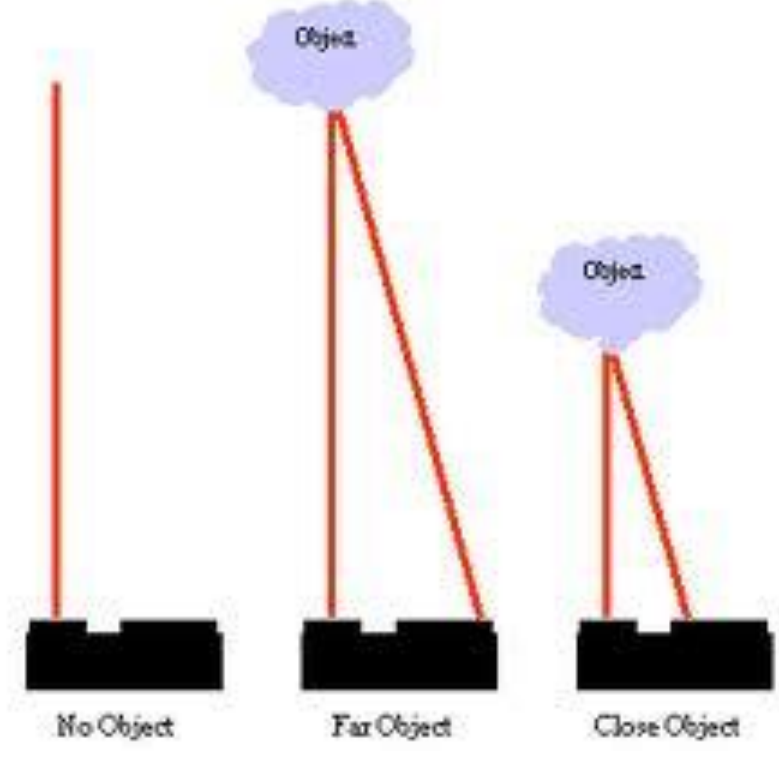
Télémètre à ultrasons SRF04



Lecture des entrées

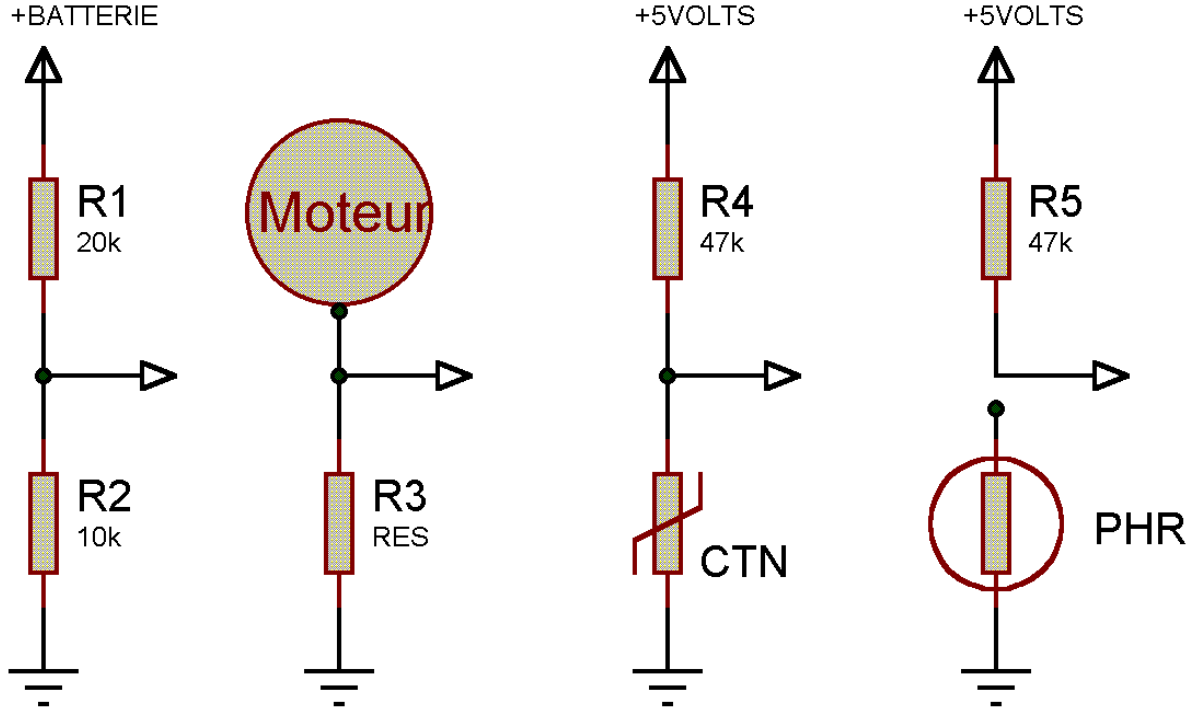
Télémètre à Infrarouge
Sharp GP2D12
sortie analogique





Lecture des entrées

Capteurs proprioceptifs



Lecture des entrées

- Logiciel de traitement
  - Pilote de capteurs
  - Mise en forme et validation
- Algorithme de traitement
  - Moyenne d'une entrée analogique sur 2^n nombres
  - Moyenne circulaire (glissante)
  - Barycentre (détection de la positon d'une ligne RSL)
  - Interpolation quadratique
  - Filtre médian (filtrage du bruit dans les images)
  - Calibration initiale ou auto-calibration en continu
  - Filtrage du 50Hz par synchronisation
  - Filtrage par FFT (µC DSP)
  - Filtre de Kalman (filtre statistique)

Algorithme de traitement

- Moyenne d'une entrée analogique sur 2^n nombres
  - Utilisation de la fonction décalage à droite comme division
  - Avantage : Simplicité du calcul

```
int   Somme;                          // Déclarations d'une variable 16bits
char   New_Data;                      // variable 8bits pour recevoir la mesure

void   Moyenne (void)
{
      for (k=0;k<8;k++)               // Faire 8 boucles
      {
         Somme = Somme + New_Data;    // Acquisition par lecture du registre
      }
      Moyenne = Somme >> 3;           // Moyenne = Somme / 8
}
```

  - Inconvénient : 2^n mesures avant d'obtenir la moyenne
  - Ne fonctionne que avec des nombres non signés

Algorithme de traitement

- Moyenne circulaire (glissante) sur 2^n nombres
  - Utilisation de la fonction décalage à droite comme division
  - Avantage : Moyenne disponible à chaque acquisition

```
int   Somme;                                    // Déclarations d'une variable 16bits
char   Data[8];                                 // Déclarations d'un tableau de 8 variables 8bits
char   New_Data;                                // variable 8bits pour recevoir la mesure

void   Moyenne (void)
{
      Somme = Somme – Data[k] + New_Data;       // Soustrait la valeur la plus ancienne et ajoute la nouvelle
       Data[k] = New_Data;                      // Mémorise la nouvelle valeur à la place de l'ancienne
      k = k +1;                                 // incrémente le compteur circulaire
      if (k == 8)
         k = 0;                                 // Boucle sur 8 valeurs
      Moyenne = Somme >> 3;                     // Moyenne = Somme / 8
}
```

  - Inconvénient : lent sur plusieurs mesures

Algorithme de traitement

- Calcul de position avec 5 capteurs de ligne
  - Calcul du barycentre
    - Chaque capteur reçoit un poids.
    - On calcule la somme des capteurs pondérés et la somme des capteurs.
    - La rapport correspond au barycentre :
    - $position = \frac{c_1+2c_2+3c_3+4c_4+5c_5}{c_1+c_2+c_3+c_4+c_5}$
  - Calcul de l'interpolation quadratique
    - Mesure de la valeur des 3 capteurs les plus forts.
    - On considérera que ces 3 valeurs représentent une courbe de type : y = $ax^2 + bx + c$, on en déduit la valeur maximale à : $x = \frac{-b}{2a}$.
    - On calcule : a $= \frac{y_1+y_3-2y_2}{2}$ et $b = y_2 - y_1 - 2ax_1 - a$

Algorithme de traitement

- Filtre médian
  - Filtre statistique impair
  - Suppression du bruit

```
uint8_t data[5];                                    // Déclarations d'un tableau de 5 variables 8bits
#define Permute(a,b)  a ^= b; b ^= a; a ^=b;
#define Tri(a,b)  if(a > b){ Permute(a,b);}

uint8_t Median (uint8_t val)
{
    data[k] = val;              // Mémorise la nouvelle valeur à la place de l'ancienne
    if (k++ == 5)               // incrémente le compteur circulaire
      k = 0;                                        // Boucle sur 5 valeurs

    Tri(data[0],data[1]);       Tri(data[1],data[2]);       Tri(data[2],data[3]);       Tri(data[3],data[4]);
    Tri(data[0],data[1]);       Tri(data[1],data[2]);       Tri(data[2],data[3]);
    Tri(data[0],data[1]);       Tri(data[1],data[2]);
    Tri(data[0],data[1]);

    return data[2];
}
```

Algorithme de traitement

- Calibration initiale et auto-calibration
  - Calibration initiale
  - Auto-calibration en continu

- Application :
  - Capteurs de ligne
  - Capteurs de couleur
  - Capteurs de distance

Algorithme de traitement

- Calibration initiale

```
struct {   uint8_t  val;                    // valeur lue
           uint8_t  min;                    // valeur minimale
           uint8_t moy;                     // valeur moyenne
           uint8_t max;                     // valeur maximale
}capt_1, capt_2, capt_3;          // déclarations des variables

void    Calibration (void)
{     capt_1.max = 0;                       // initialiser les valeurs max à 0
      capt_1.min = 255;                     // initialiser les valeurs min à 255

      // Réaliser la mesure des valeurs des capteurs sur l'étendue du domaine
      capt_1.val = ADC_read (capt_1);
      if (capt_1.val < capt_1.min)
         capt_1.min = capt_1;
      if (capt_1.val > capt_1.max)
         capt_1.max = capt_1;

      // Calcul de la valeur moyenne
      capt_1.moy = (capt_1.max + capt_1.min) >>2;
      // Calcul de la valeur calibrée sur 100
      capt_1.val = ((capt1_val – capt_1.min)*100)/(capt_1.max - capt_1.min);
}
```

Actionneurs – Liaison bidirectionnelle

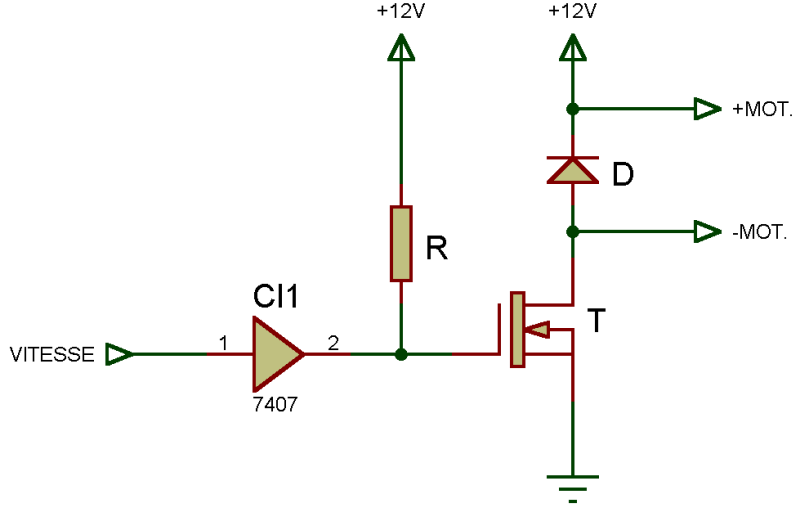
- Nature des sorties
  - Logique
  - Analogique (CNA, MLI avec filtre passe-bas)
  - MLI (PWM)
- Ressources enfouies
  - Diverses techniques de commande d'actionneurs
  - Timer
- Liaison bidirectionnelle
  - Liaison ponctuelle avec une interface intelligente
  - Liaison sans fil (IR, ZigBee, Bluetooth, wifi)
  - Liaison entre systèmes autonomes ou avec un système centralisé

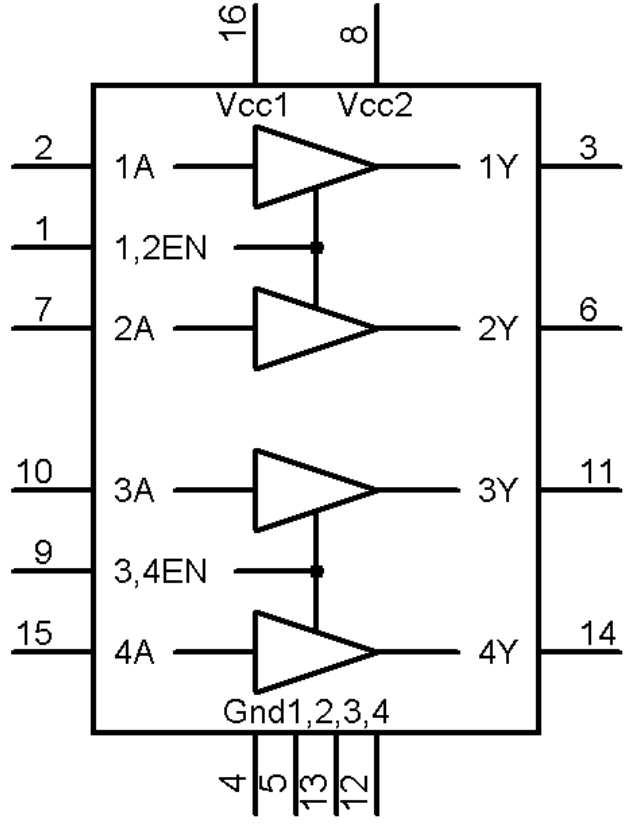
Les divers types de moteurs CC

- Les moteurs CC à aimant permanent
- Commande par MLI

Les divers types de moteurs CC

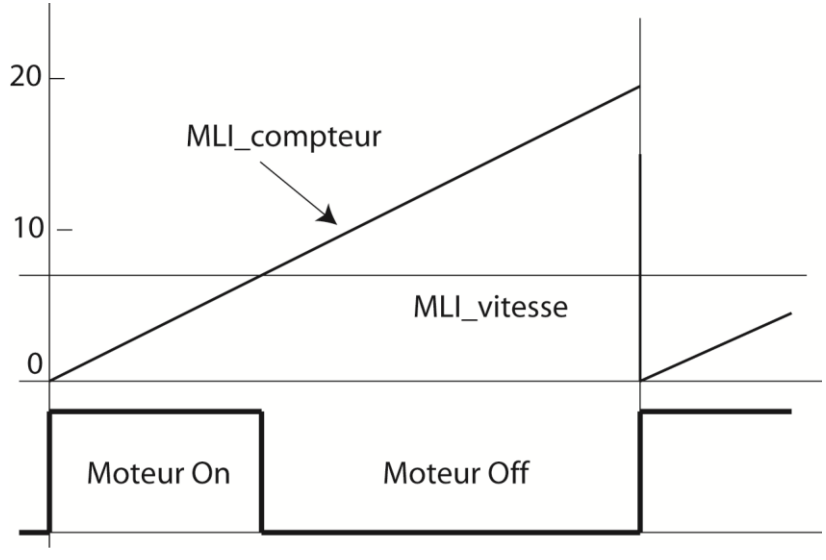
- Les moteurs CC à aimant permanent
- Commande par MLI

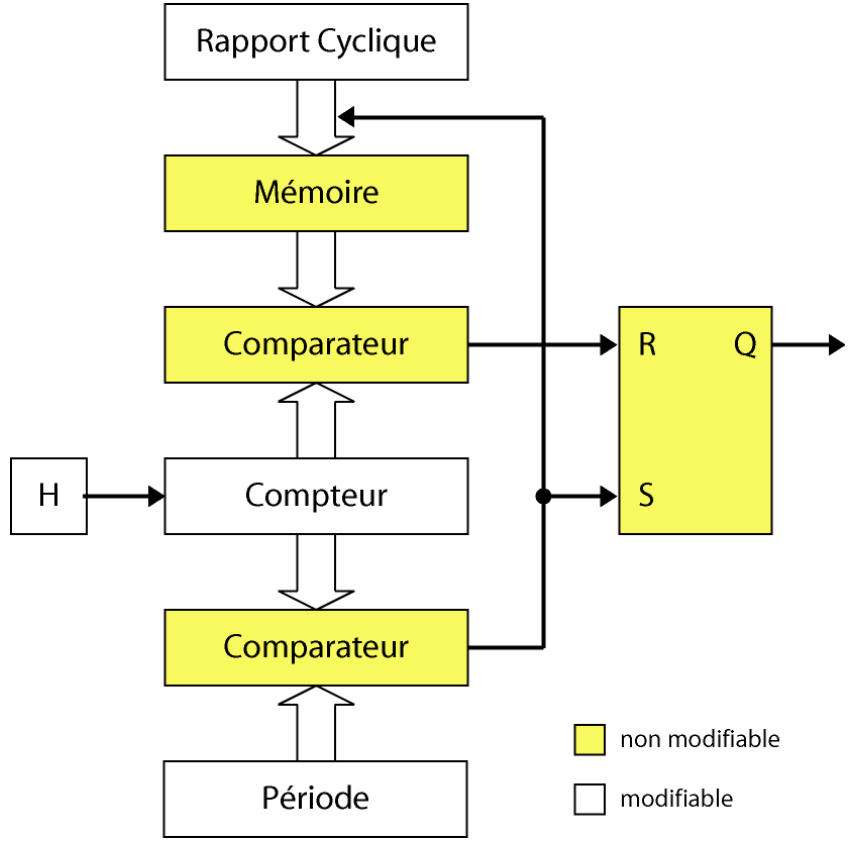




Les divers types de moteurs CC

- Les moteurs CC à aimant permanent
- Commande par MLI



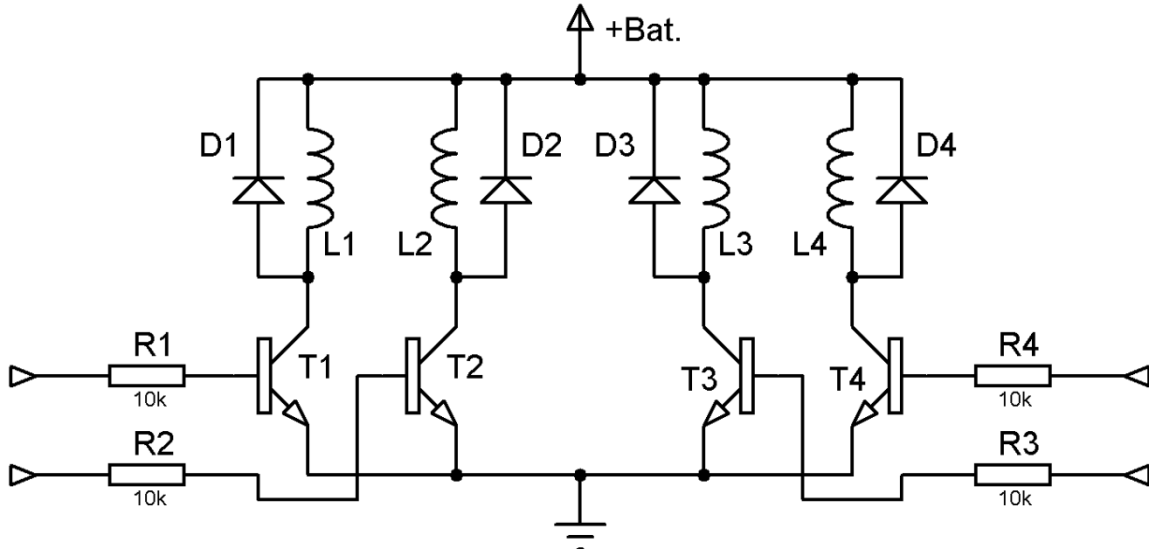


Les divers types de moteurs CC

- Les moteurs pas à pas
- Commande par lecture d'une table des valeurs

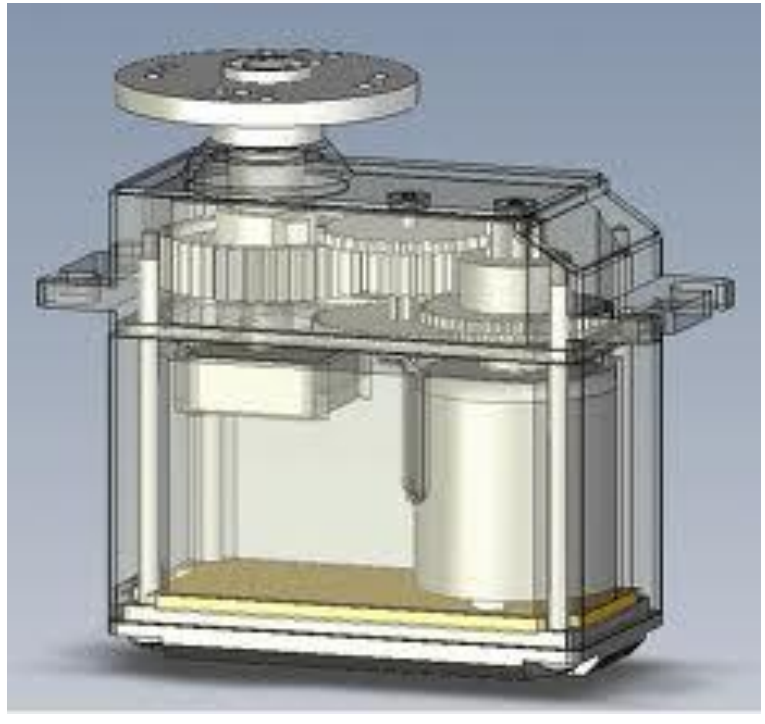
Les divers types de moteurs CC

- Les moteurs pas à pas
- Commande par lecture d'une table des valeurs



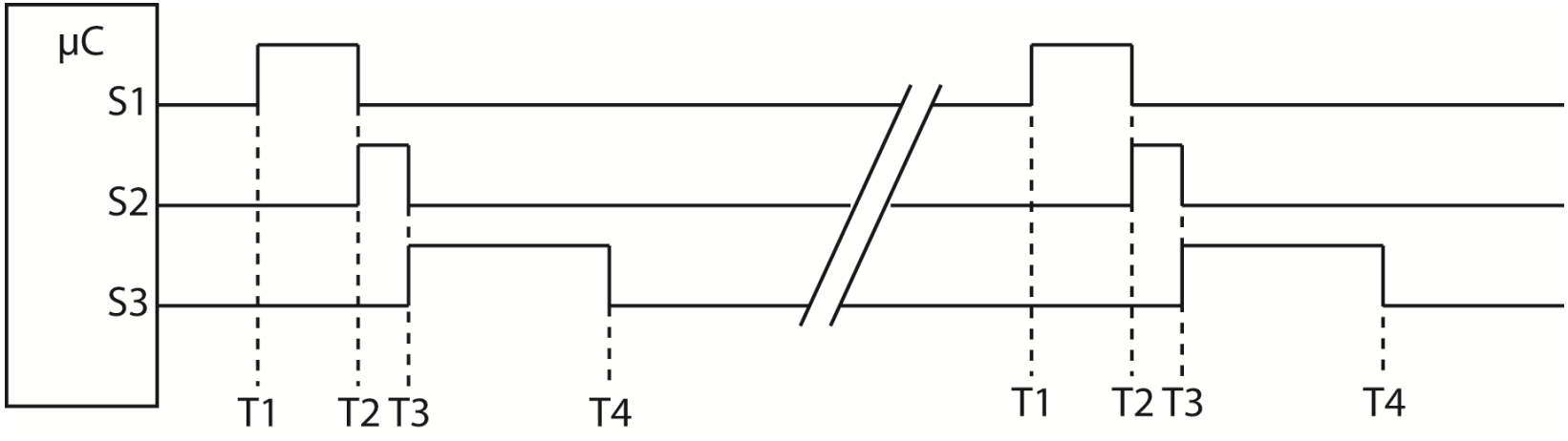
Les divers types de moteurs CC

- Les servo moteurs
- Pas d'interface de puissance
- Commande par MLI sur interruption



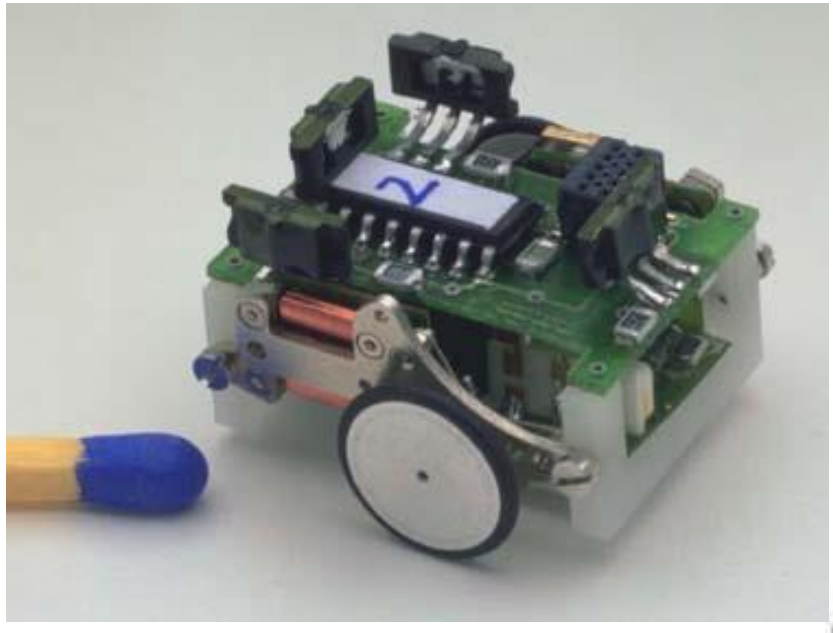
Les divers types de moteurs CC

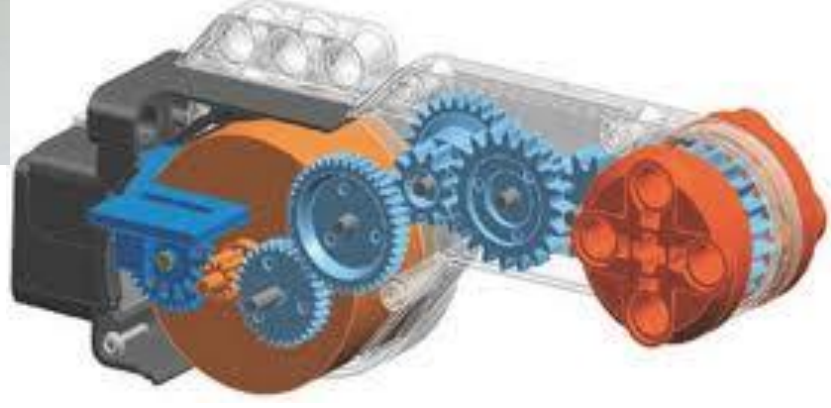
- Les servo moteurs
- Commande par MLI sur interruption



Les divers types de moteurs CC

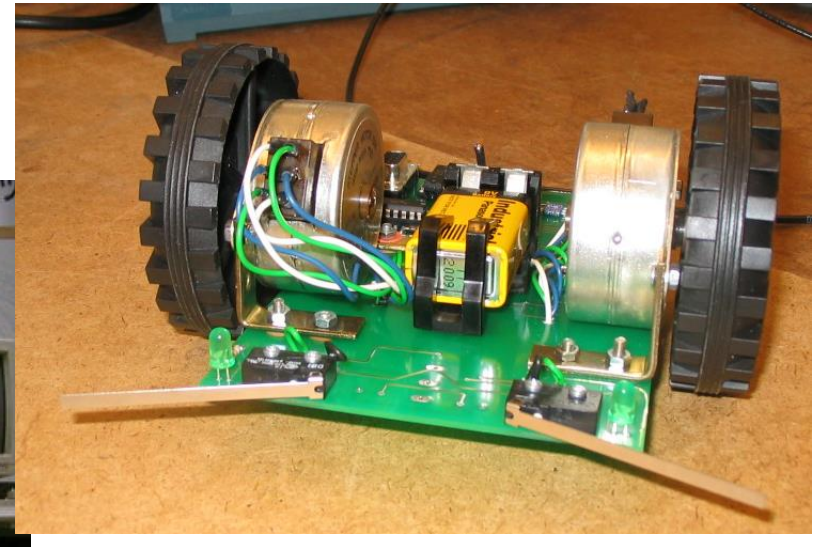
- Essayez de reconnaitre ces types de moteurs !





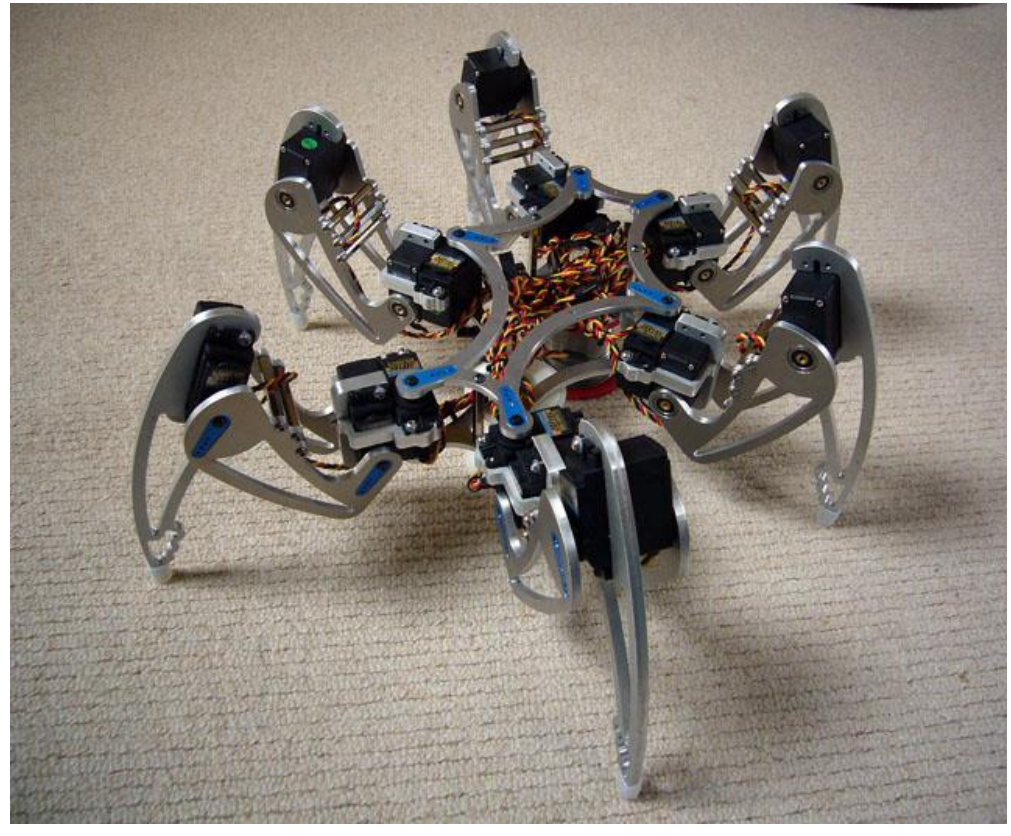
Les divers types de moteurs CC

- Essayer de reconnaitre ces types de moteurs !



Les divers types de moteurs CC

- Essayer de reconnaitre ces types de moteurs !



Les divers types de moteurs CC

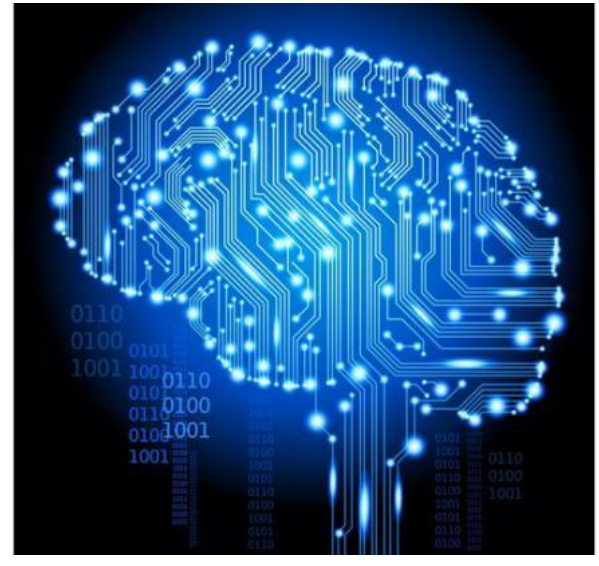
- Essayer de reconnaitre ces types de moteurs !

Quelques vidéos

- Boston Dynamics
- Korea Humanoid Robot
- Adam's Savage Spider
- Micromouse

Intelligence Artificielle

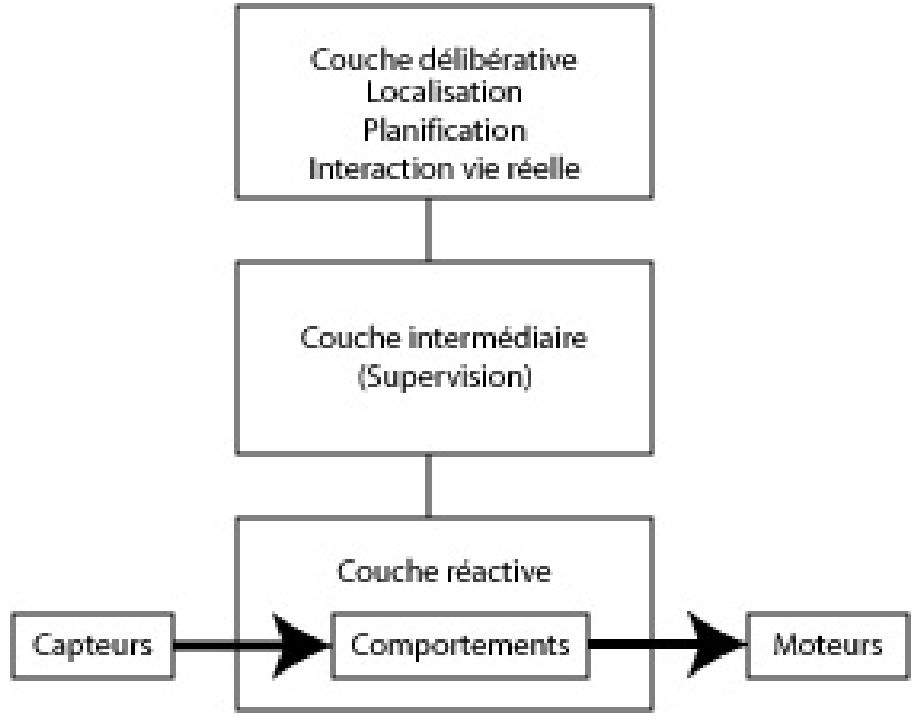
- Architecture logicielle
- Architecture à interactions prioritaires
- Correcteurs
- Algorithmes génétiques
- Réseaux de neurones
- Systèmes expert



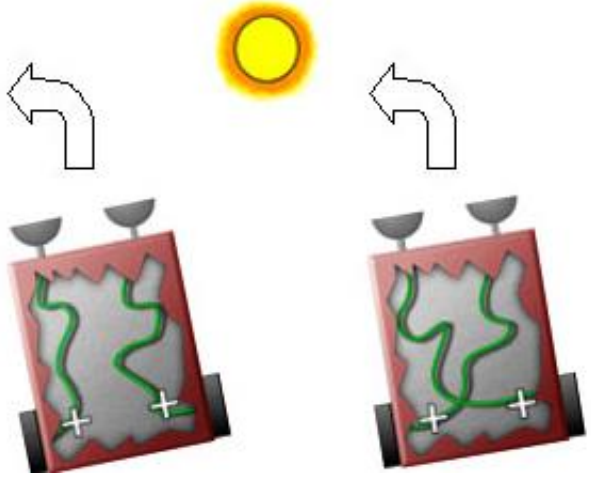
Architecture logicielle

- L'approche traditionnelle

- Architecture hybride



- Architecture réactive

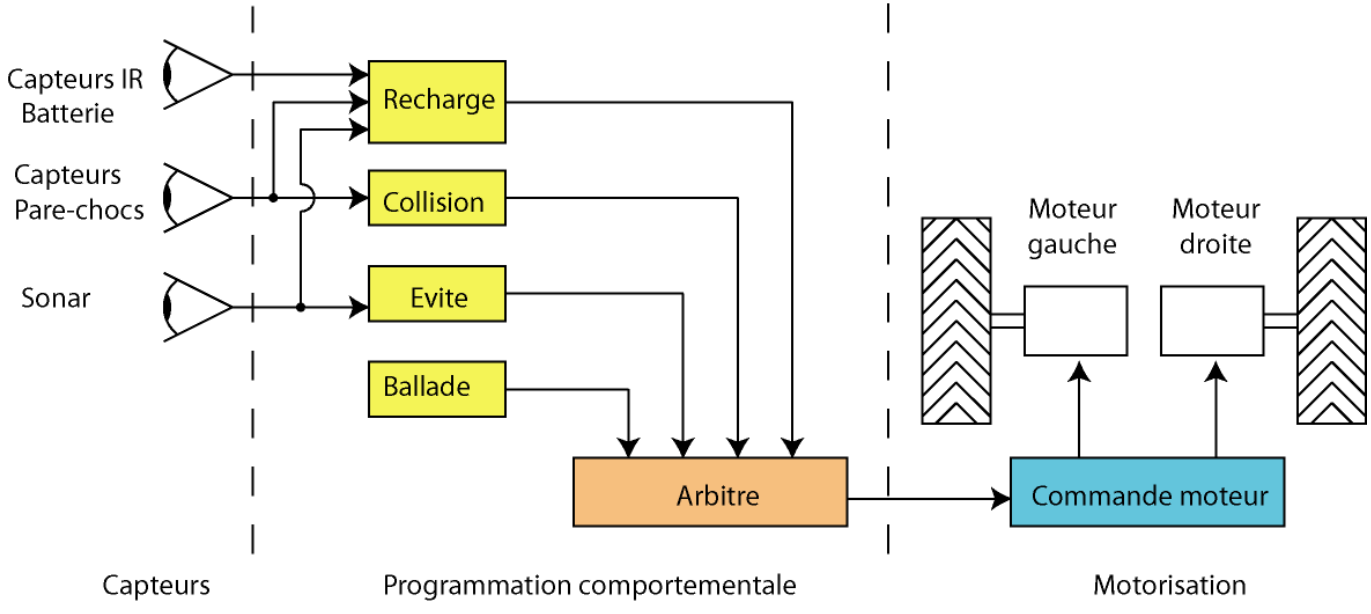


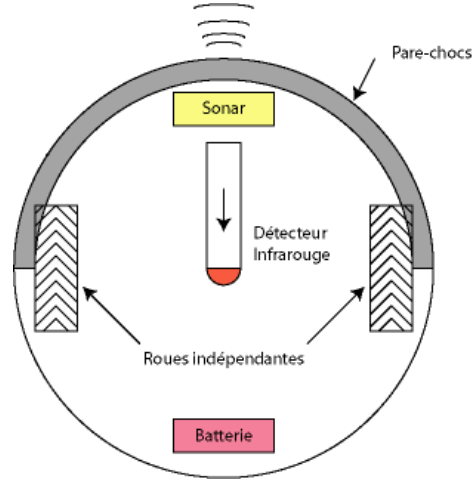
Architecture à interactions prioritaires

- Exemple : Roomba, le robot aspirateur de la société ***« Irobot »***
  - Pare-chocs : détecte une collision
  - Télémètre infrarouge : mesure une distance
  - Détecteur infrarouge : détecte une balise infrarouge
  - Batterie : mesure la tension de la batterie
  - Moteurs : contrôle la vitesse des moteurs

Architecture à interactions prioritaires

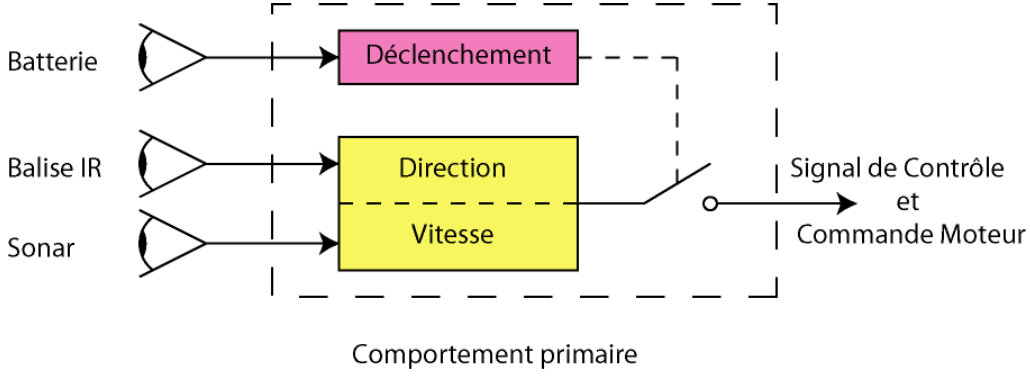
- Programmation comportementale

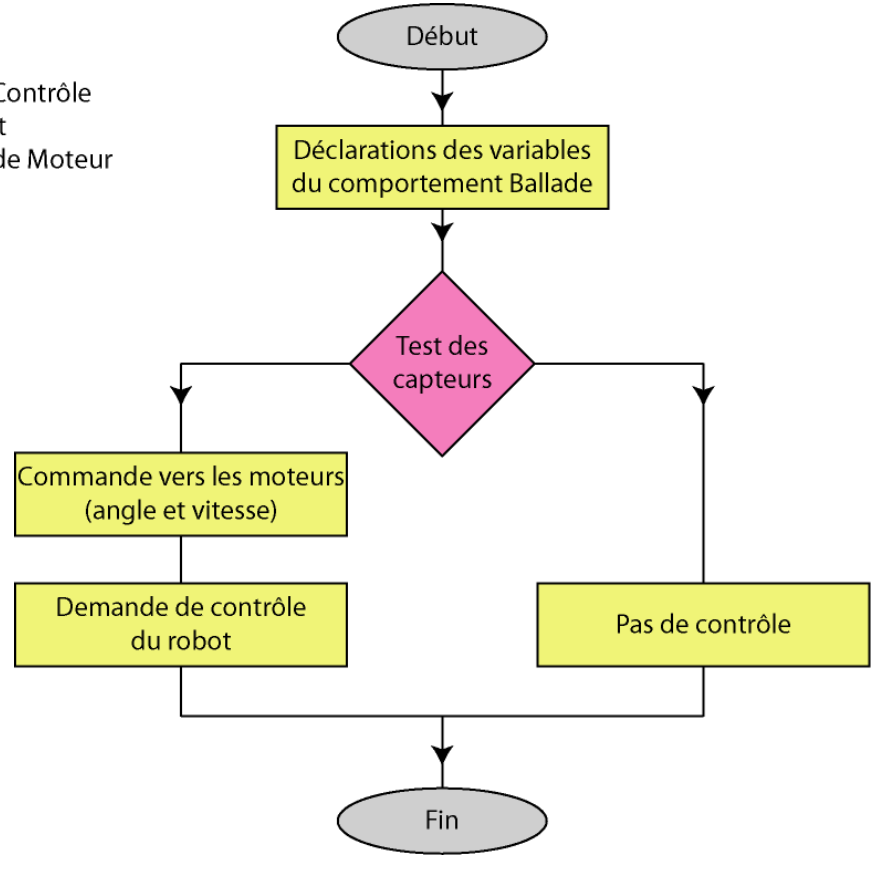




Architecture à interactions prioritaires

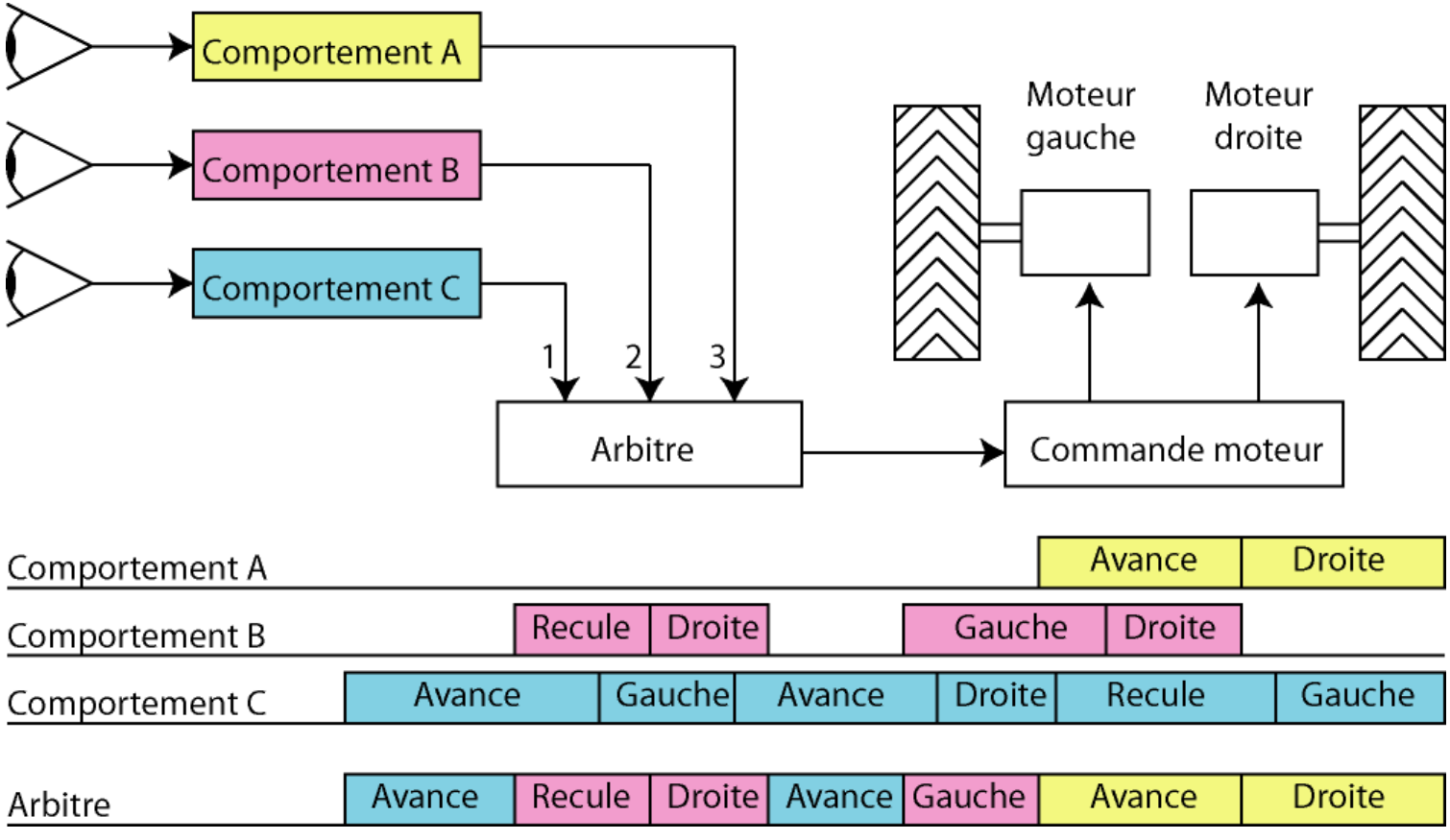
- Les comportements





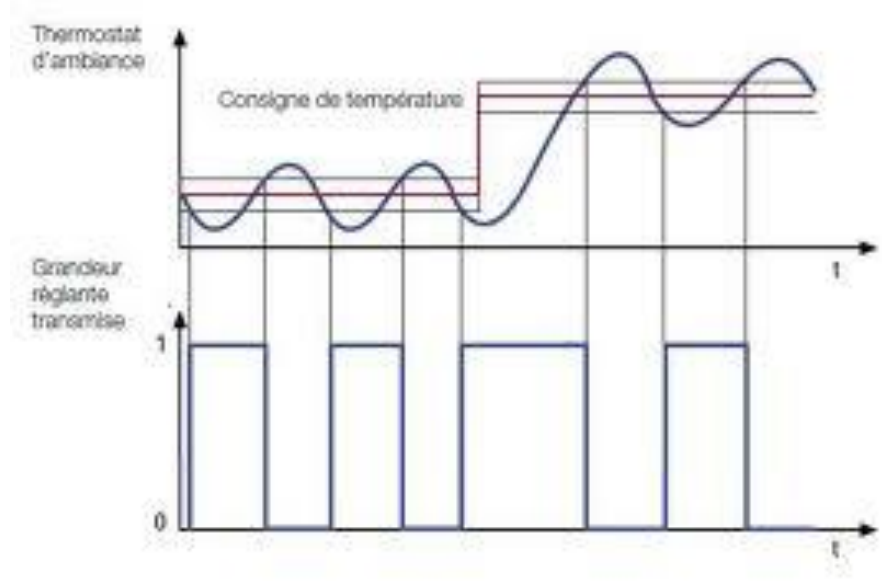
Architecture à interactions prioritaires

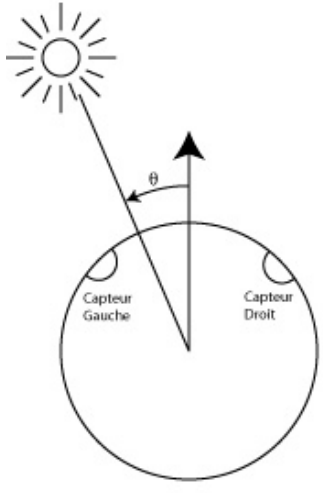
- L'arbitrage

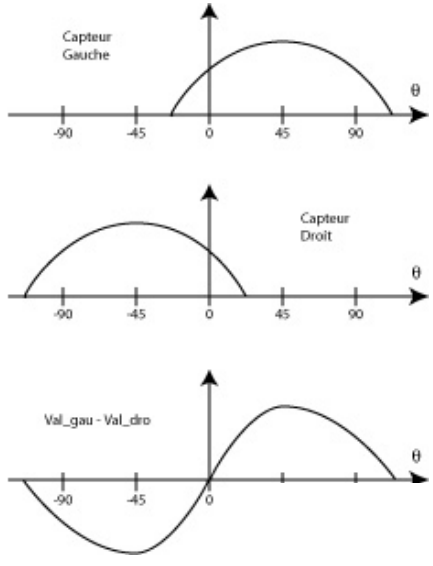


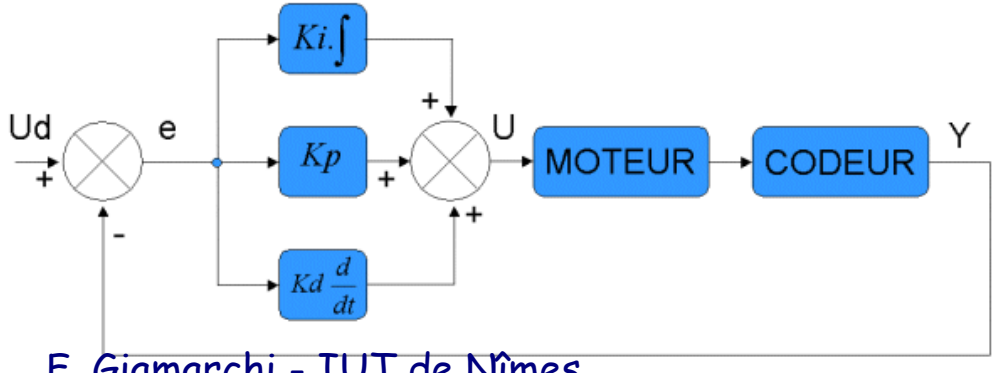
Les correcteurs

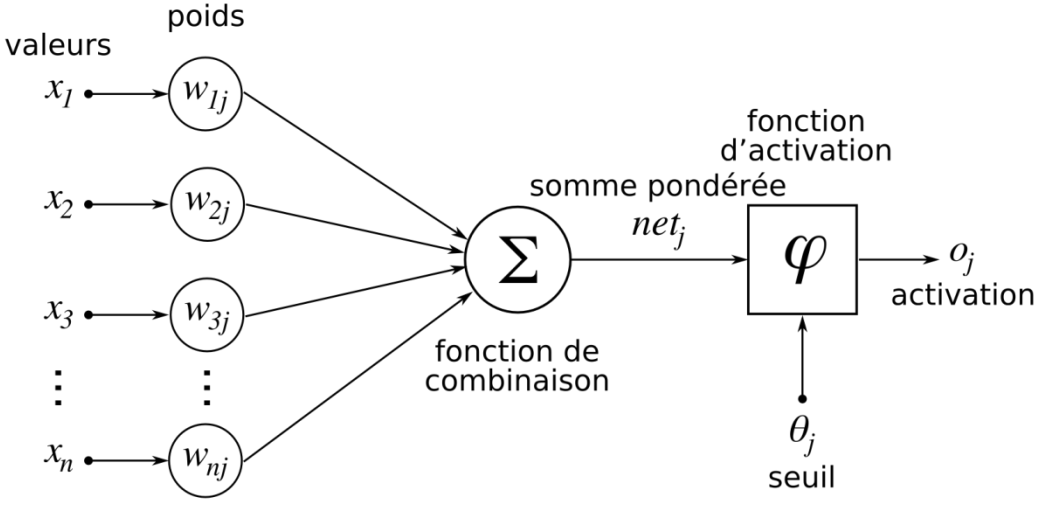
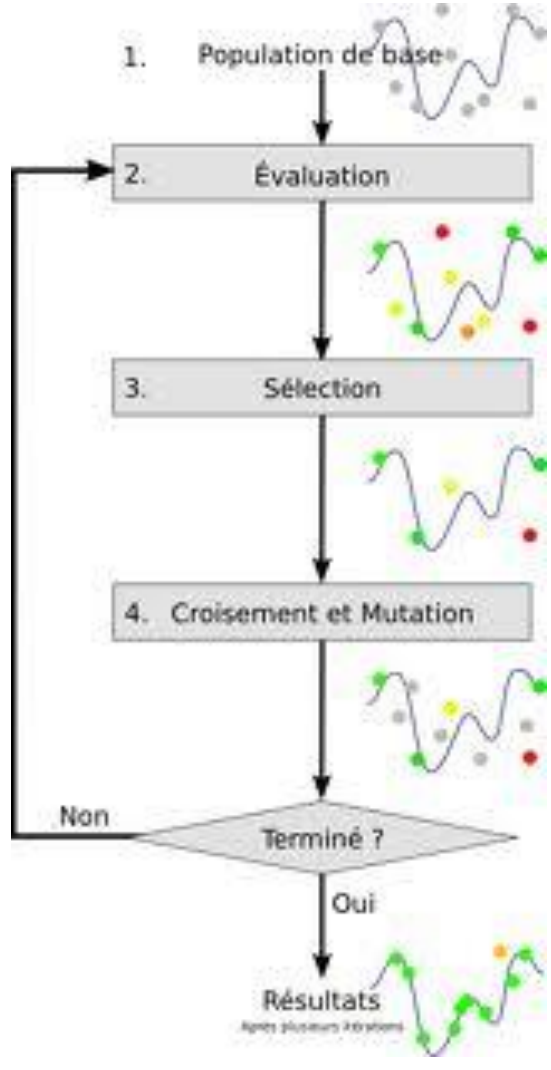
- Correcteur tout ou rien
- Correcteur proportionnel
- Correcteur PID









L'évolution de la programmation en robotique

- Algorithmes génétiques
- Réseaux de neurones
- Systèmes expert