Robots Mindstorms et mathématiques au Lycée

Matthieu Martel
DALI - Université de Perpignan Via Domitia
LIRMM : CNRS & Université Montpellier 2
matthieu.martel@univ-perp.fr

Université d'été de Mathématiques de Sourdun

Août 2011

UPVD
Université de Perpignan Via Domitia

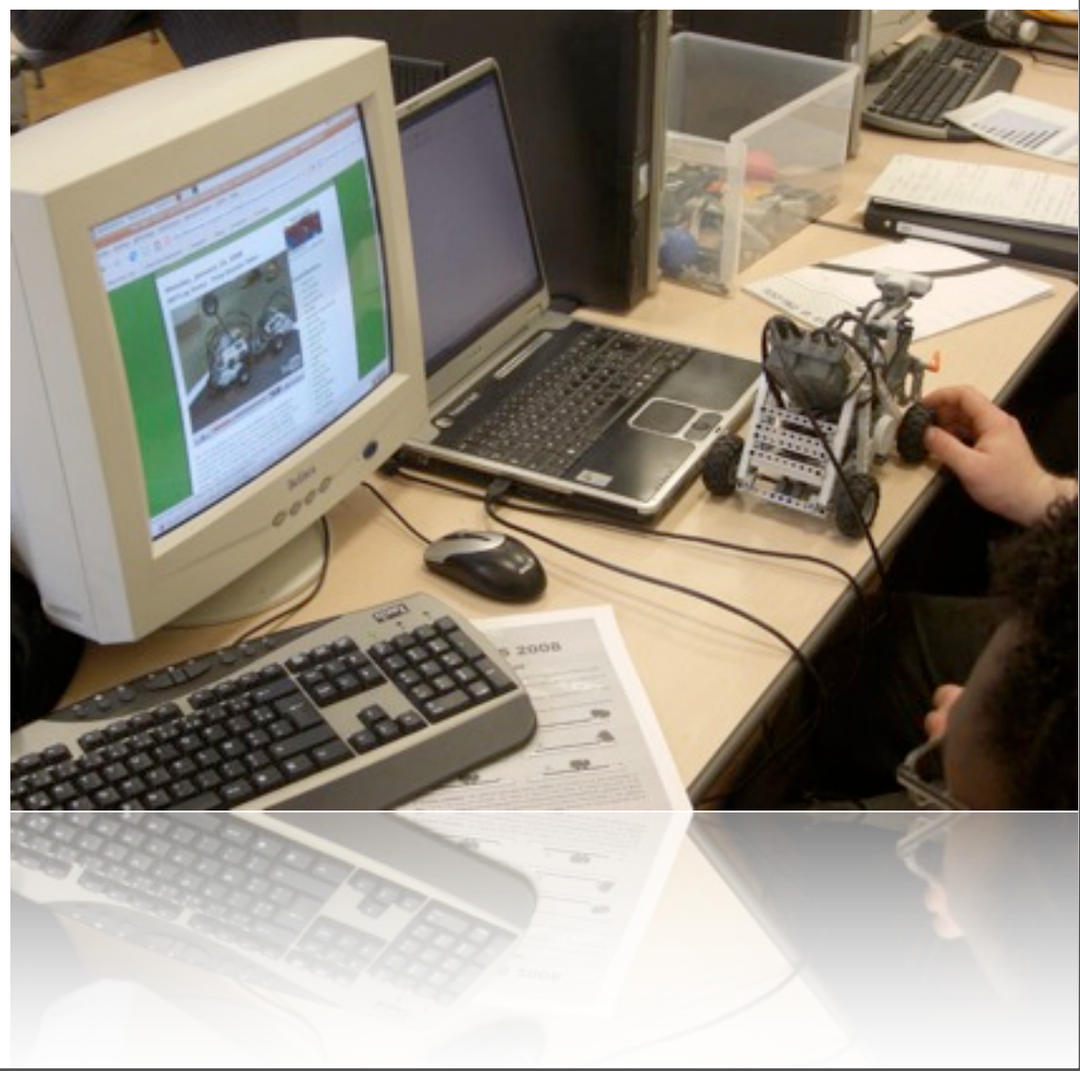
LIRMM
Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier

Plan

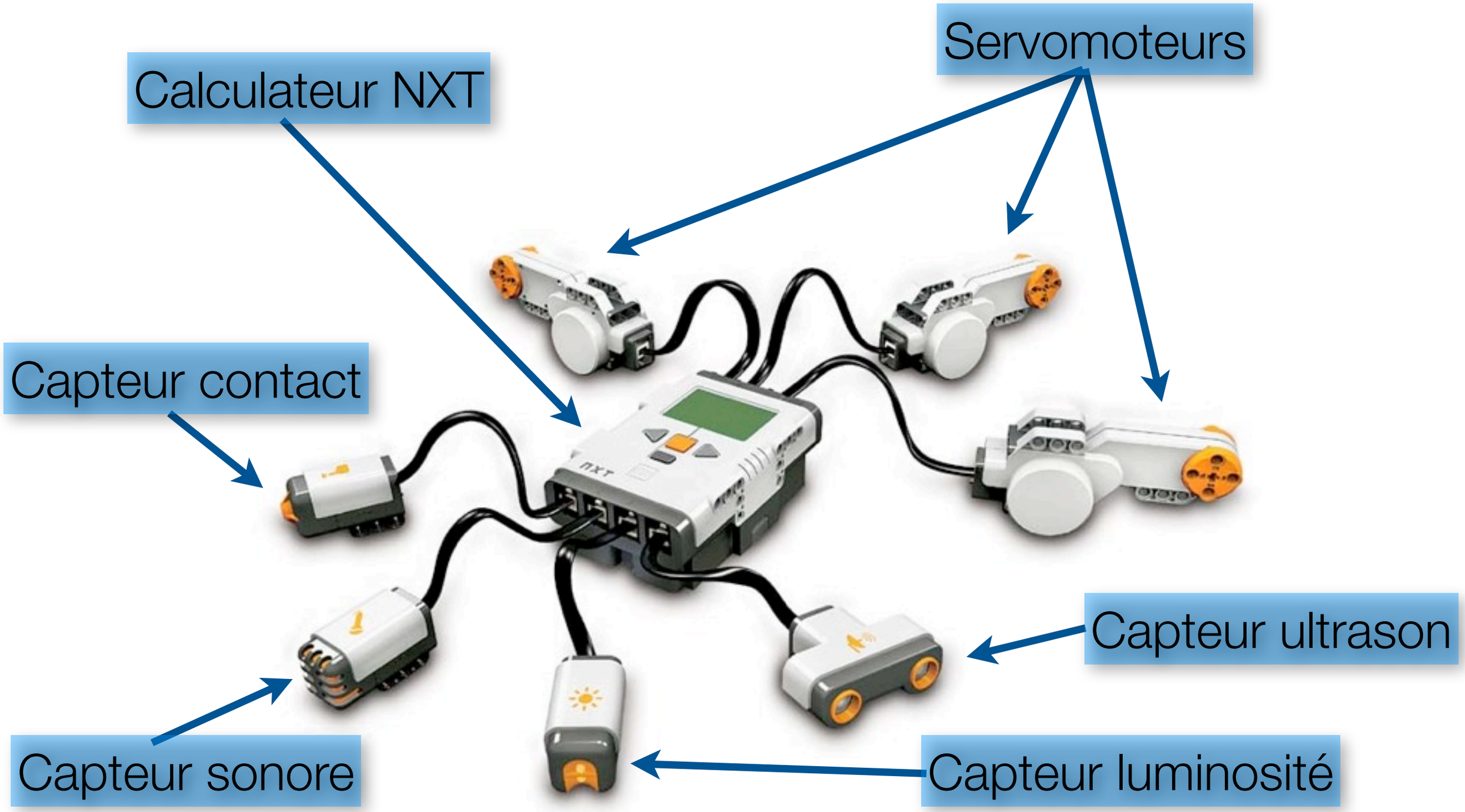
- Introduction
- Navigation et localisation
- Autres problèmes
- Conclusion

Plan

- **Introduction**
- Navigation et localisation
- Autres problèmes
- Conclusion



Architecture générale



Capteurs et moteurs

Contact

mesure la pression exercée sur l’embout

Sonore

mesure le niveau sonore (dB)

Luminosité

mesure l’intensité lumineuse

Ultrasons

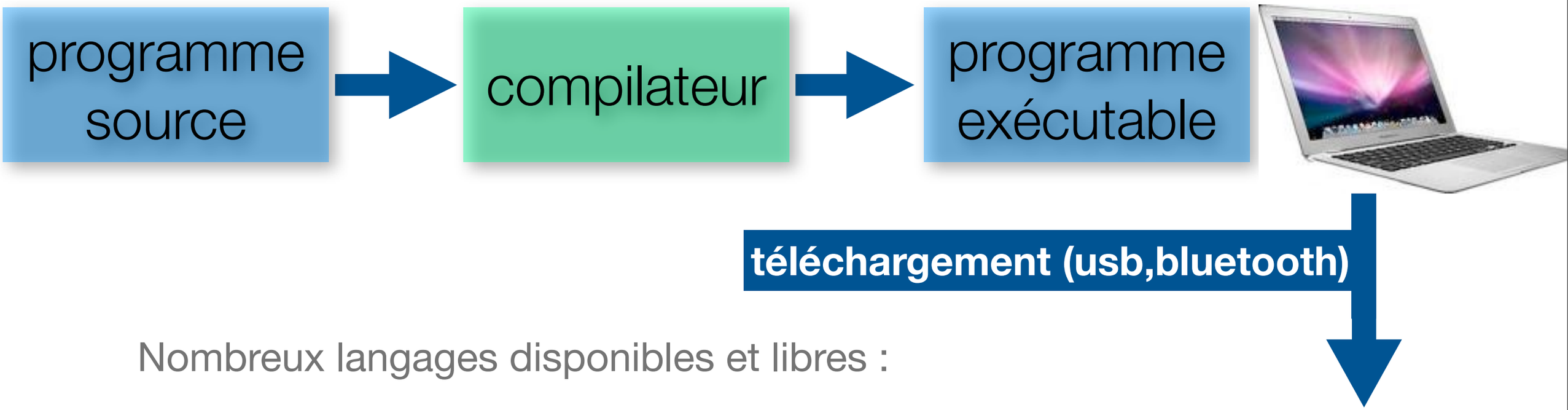
mesure la distance à un obstacle

Couleur

mesure la couleur

- Moteurs munis de capteurs de rotation
- Permet l’asservissement de 2 moteurs reliés à 2 roues différentes

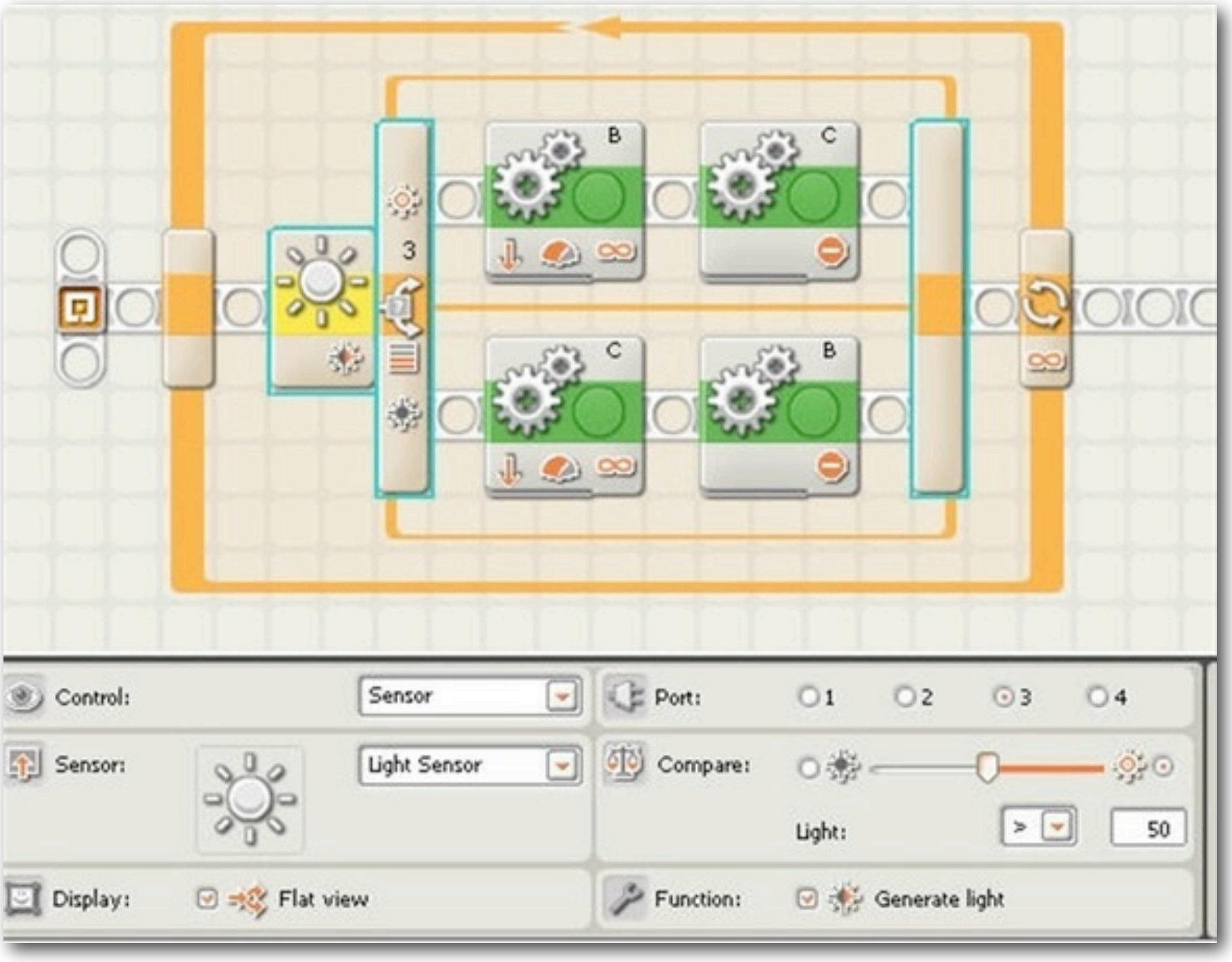
Programmation



Nombreux langages disponibles et libres :

- Langage C (NXC, RobotC)
- Java (Lejos)
- Assembleur/bytecode (NBC)
- Langages graphiques (NXT-G, Robolab)
- Langage réactif (Lejos OSEK)
- etc.

Exemple : programme NXT-G



Exemple : programme NXC

```
OnRevReg(OUT_BC,100,OUT_REGMODE_SYNC);  // démarrage des moteurs

// boucle interactive de contrôle
while(Sensor(IN_1)==0)        // tant que l'on ne rencontre pas d'obstacle
{
    // Etape 1 : mesure de la distance au mur
    d = SensorUS(IN_4);       //   d contient la distance mesurée

    // Etape 2 : calcul de la réponse : angle a à donner au volant
        if (d<D_MIN) {        //   si le robot est trop près du mur
        reponse = 30;         //     volant vers la gauche
    } else {                  //   sinon
        if (d>D_MAX) {        //     si le robot est trop loin du mur
            reponse = -30;    //       volant vers la droite
        } else {              //     sinon
            reponse = 0;      //       volant droit
        };
    };

    // Etape 3 : action sur le volant, modifie l'évolution de d
    RotateMotor(OUT_A,100,reponse-angle); // braquage du volant en fonction de a
    angle = reponse;                      // nouvel angle du volant;

    // Etape 4 : déplacement du robot
    Wait(100);                 // laisser le robot avancer pendant 100ms
```

Projet 1 : Combats de sumos (mai 2008)

Projet 2 : Morpion (décembre 2010)

Projet 3 : Suivi de mur (décembre 2009)

Projet 4 : Biathlon (mai 2010)

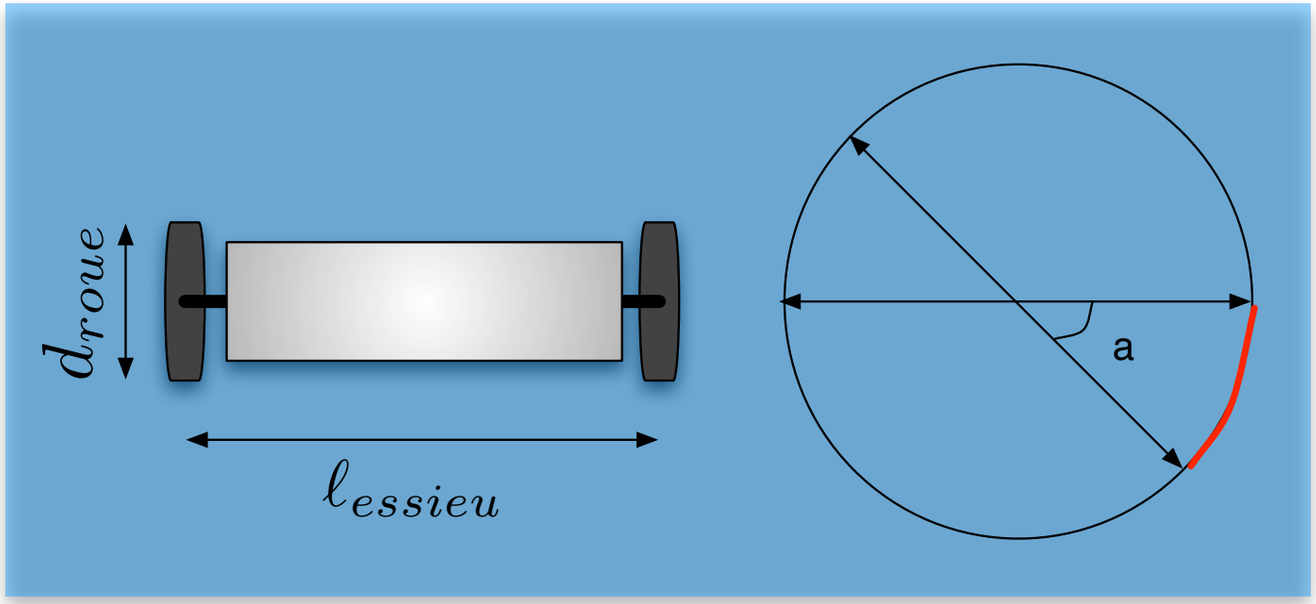
Projet 5 : Metro (concours Alstom, avril 2009)

Plan

- Introduction
- **Navigation et localisation**
- Autres problèmes
- Conclusion

Navigation élémentaire

Virer d'un angle *a* :



- parcourir distance :

$$d = \frac{\pi \times \ell_{essieu} \times a}{360}$$

- tourner les roues de l'angle :

$$\alpha = \frac{360 \times d}{\pi \times d_{roue}} = \frac{a \times \ell_{essieu}}{d_{roue}}$$

360	$\pi \times \ell_{essieu}$
a	d

360	$\pi \times d_{roue}$
α	d

Angle pour avancer d'une distance *d* :

360	$\pi \times d_{roue}$
θ	d

$$\theta = \frac{360 \times d}{\pi \times d_{roue}}$$

Navigation : implémentation

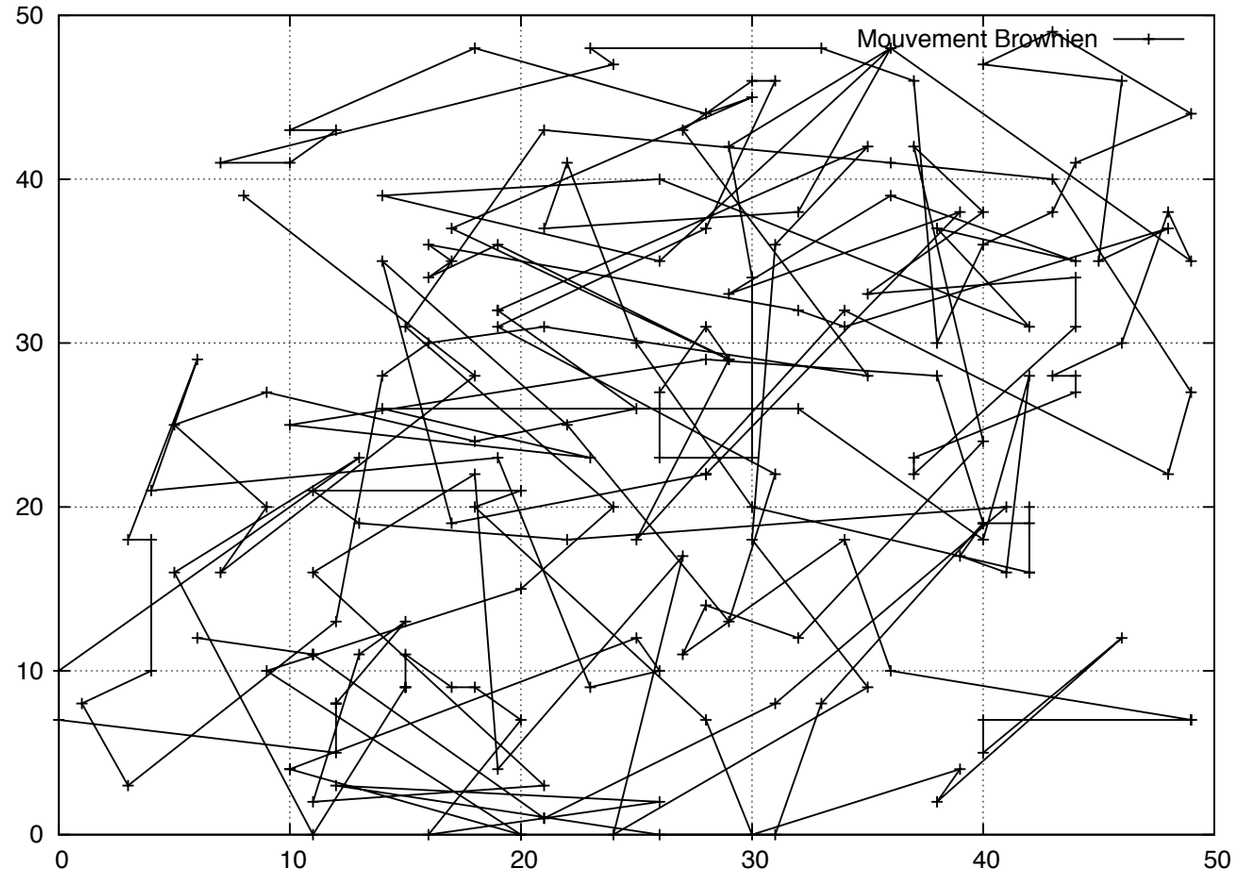
```
#define ESSIEU 11.2             // longueur de l'essieu en cm
#define DIAMETRE_ROUE 5.5       // diamètre des roues en cm
```

```
// fonction permettant au robot de tourner d'un angle a (en degrés)
inline void virer(float a) {
    float angle_rot;

    angle_rot = a*ESSIEU/DIAMETRE_ROUE;
    RotateMotor(OUT_B,75,angle_rot);
    RotateMotor(OUT_C,75,-angle_rot);
}
```

```
// fonction permettant d'avancer tout droit de d cm
inline void avancer(float d) {
    float angle_dist;

    angle_dist = (360*d)/(PI*DIAMETRE_ROUE);
    RotateMotor(OUT_BC,75,angle_dist);
}
```

Couverture d’un espace

- **Mouvement Brownien :**
  - Chaque pas a une longueur et une direction aléatoire
  - En dimension 2, la probabilité qu’un point soit visité tend vers 1 lorsque le nombre de pas tend vers l’infini
  - Arrêt du robot lorsque toutes les cases de la grille ont été visitées



Mouvement Brownien : implémentation

```
// calcul de la prochaine position du robot
x_prime=-1; y_prime=-1;
while // tant que l'on ne choisit pas un pas valide
      (! ((x_prime<N*PAS)&&(x_prime>0)&&
           (y_prime<N*PAS)&&(y_prime>0))
      )
{
     r=Random(DIST_MAX);         // longueur du pas
     theta=Random(ANGLE_MAX);  // angle du virage
     dir_prime=dir+theta;        // direction du robot
     if (dir_prime>=360) { dir_prime=dir_prime-360; };
     x_prime=x+r*cosd(dir_prime);     // nouvelle position
     y_prime=y+r*sind(dir_prime);

 };

 // déplacement du robot
virer(theta);
avancer(r);
```

Arrêt lorsque toutes les cases de la grille ont été visitées

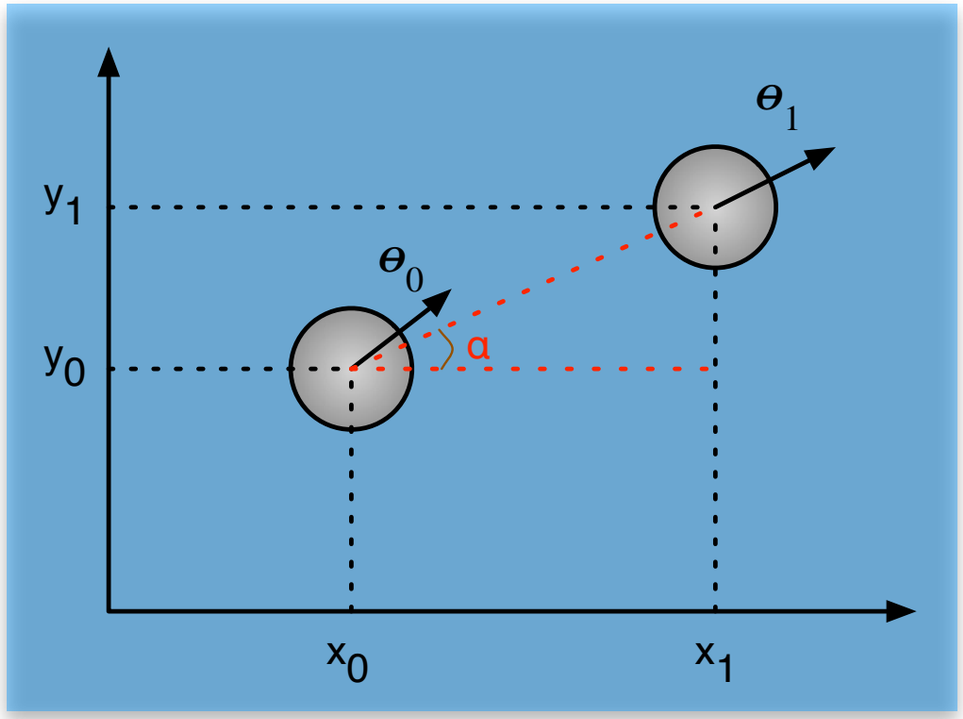
Déplacement suivant une courbe paramétrée

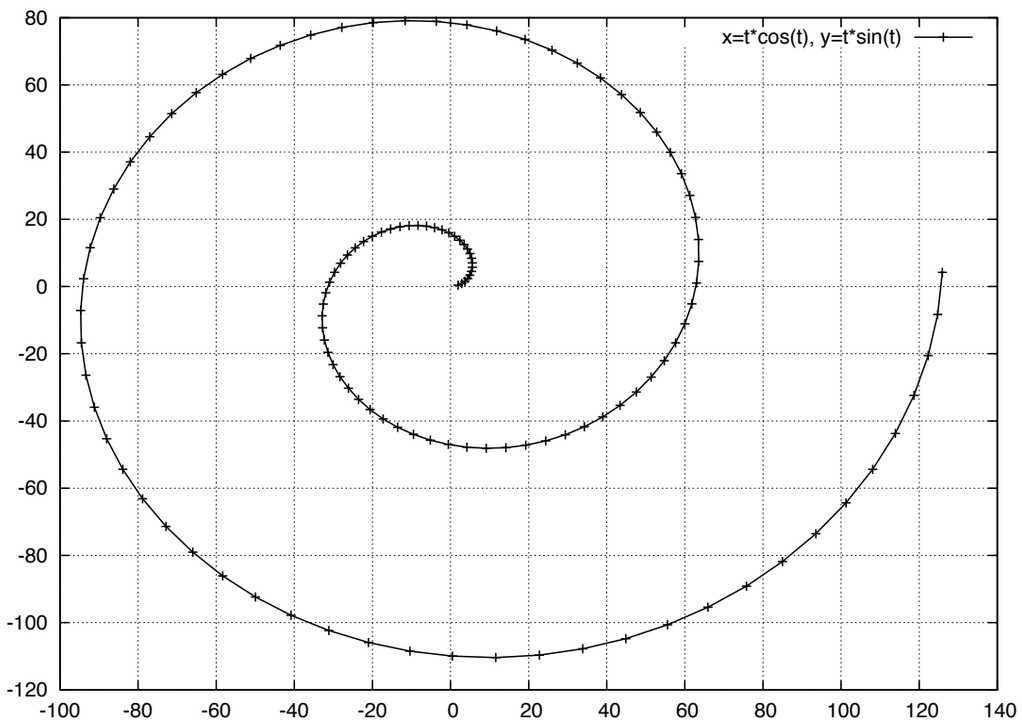
$$x_0 = f(t) \qquad y_0 = g(t)$$
$$x_1 = f(t + \Delta t) \qquad y_1 = g(t + \Delta t)$$

$$d = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$$

$$\alpha = atan\left(\frac{y_1 - y_0}{x_1 - x_0}\right)$$

$$\theta_1 = \alpha - \theta_0$$



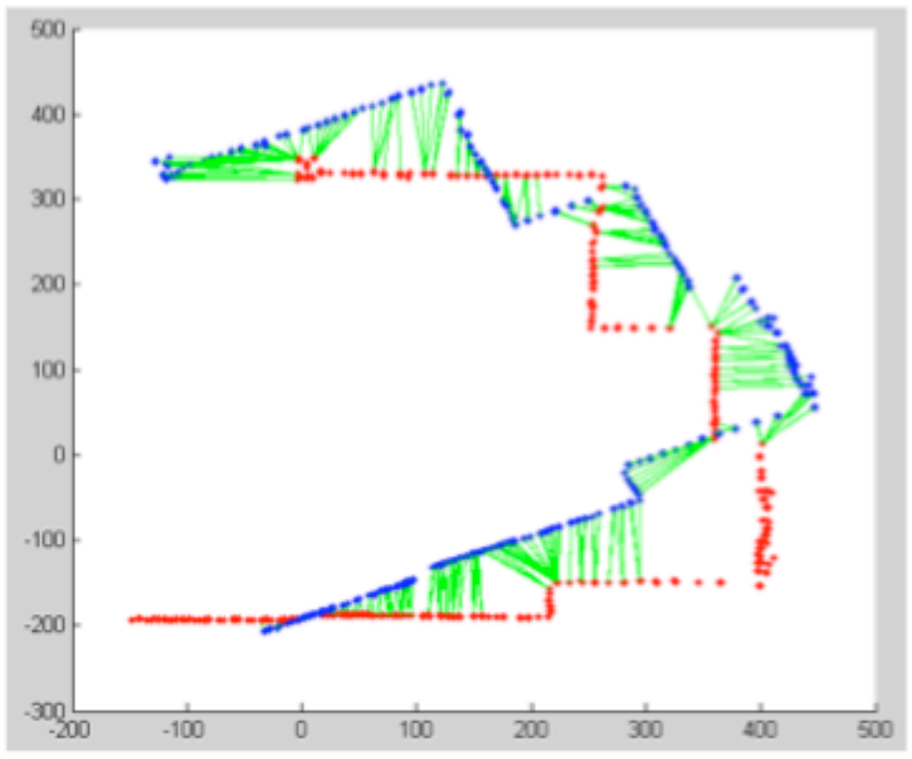


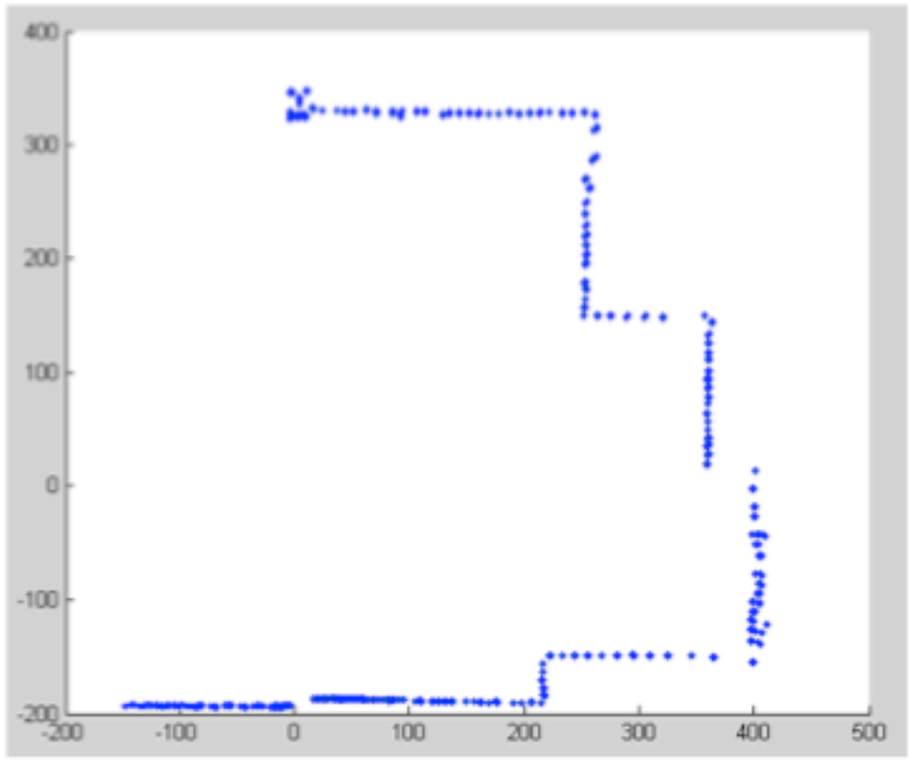
Courbe paramétrée : implémentation

```
// boucle principale
while(t<8.0*PI)                                          // pour 4 tours
{
    t = t+dt;                                            // t = instant suivant
    x1 = f(t); y1 = g(t);                                // (x1,y1) point suivant

    // calcul des nouvelles distance et direction
    d = sqrt((x1-x0)*(x1-x0)+(y1-y0)*(y1-y0));           // d en cm
    a1 = atan2d(y1-y0,x1-x0) ;                           // a1 direction en degrés
    a=a1-a0;                                             // virer de a degrés
    if (a<-180.0) a=a+360; else if (a>180.0) a=a-360;    // normalisation de a

    // déplacement du robot
    virer(a);
    avancer(d);

    x0=x1; y0=y1; a0=a1;                                 // màj de la position
}
```

Localisation

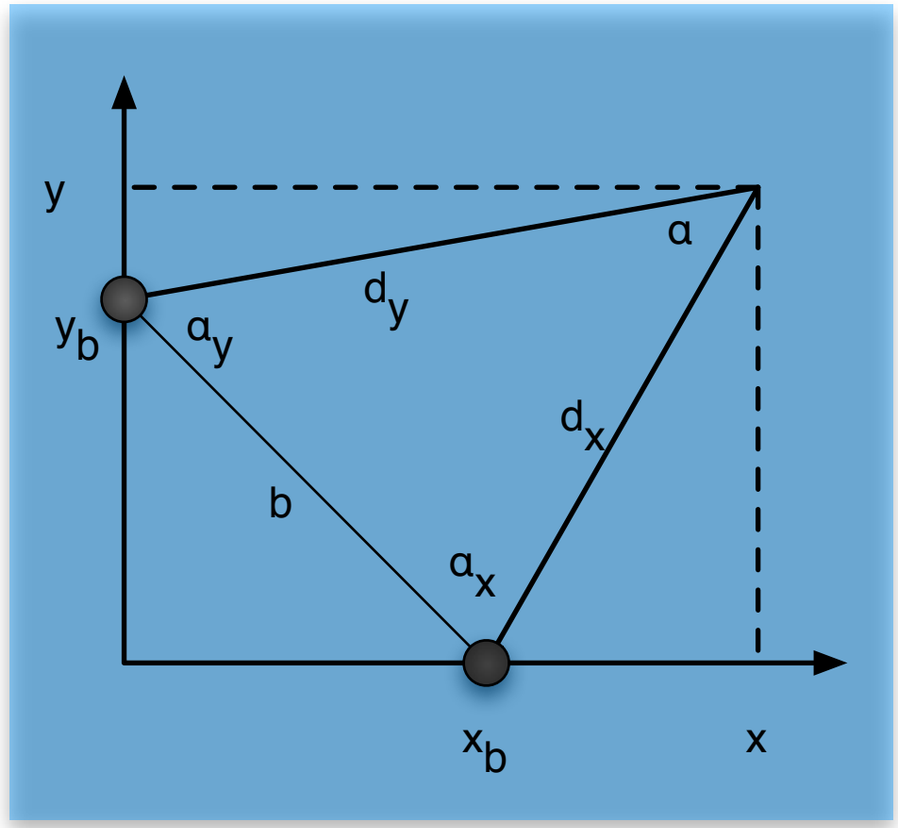
- Odométrie : estimation de la position à partir des mouvements des roues
- Localisation par perception :
  - Balises
  - Cameras, cartographie





Positionnement à l'aide de balises

- Positionnement à l'aide de deux balises :
- Hypothèses :
  - Repère orthonormé
  - Balises en $(x_b,0)$ et $(0,y_b)$ avec $x_b = y_b$



$$\frac{b}{sin\ \alpha} = \frac{d_y}{sin\ \alpha_x} = \frac{d_x}{sin\ \alpha_y}$$

$$\alpha_x = \sin^{-1}\left(\frac{d_y\ sin\ \alpha}{b}\right) \qquad \alpha_y = \sin^{-1}\left(\frac{d_x\ sin\ \alpha}{b}\right)$$

$$x = x_b + sin\left(\alpha_x - \frac{\pi}{4}\right) \qquad y = y_b + sin\left(\alpha_y - \frac{\pi}{4}\right)$$

Positionnement : implémentation (1)

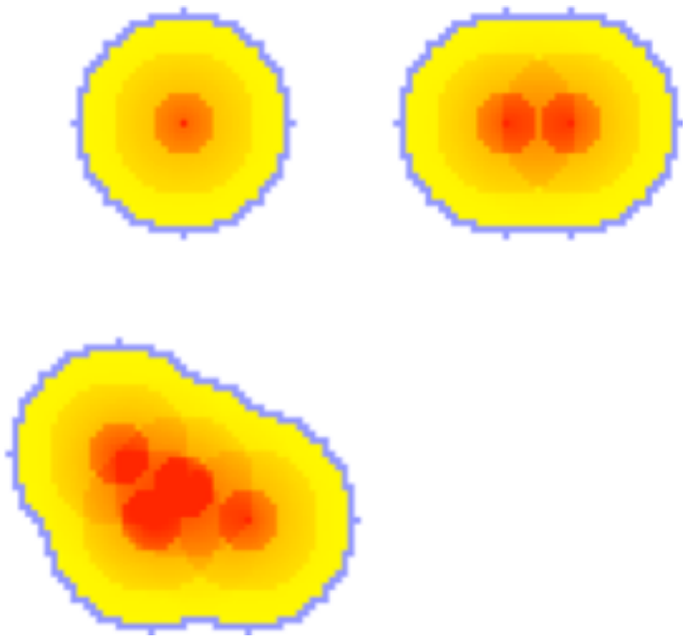
```
sub detecter_balise() {
    d = DIST_MAX+1;
    theta=0;
    while(d>DIST_MAX) { // tant que l'on ne voit pas la balise
        virer(DELTA_ANGLE);
        d=SensorUS(IN_4);                    // mesurer distance
        theta=theta+DELTA_ANGLE;             // màj angle
    };
    d_prime=d; // tant que l'on se rapproche de la balise en tournant
    while(d<=d_prime) {
        virer(DELTA_ANGLE);
        d=SensorUS(IN_4);                    // mesurer distance
        theta=theta+DELTA_ANGLE;             // màj angle

    };
    virer(-DELTA_ANGLE);
    theta=theta-DELTA_ANGLE;                 // angle donnant la distance min
}
```

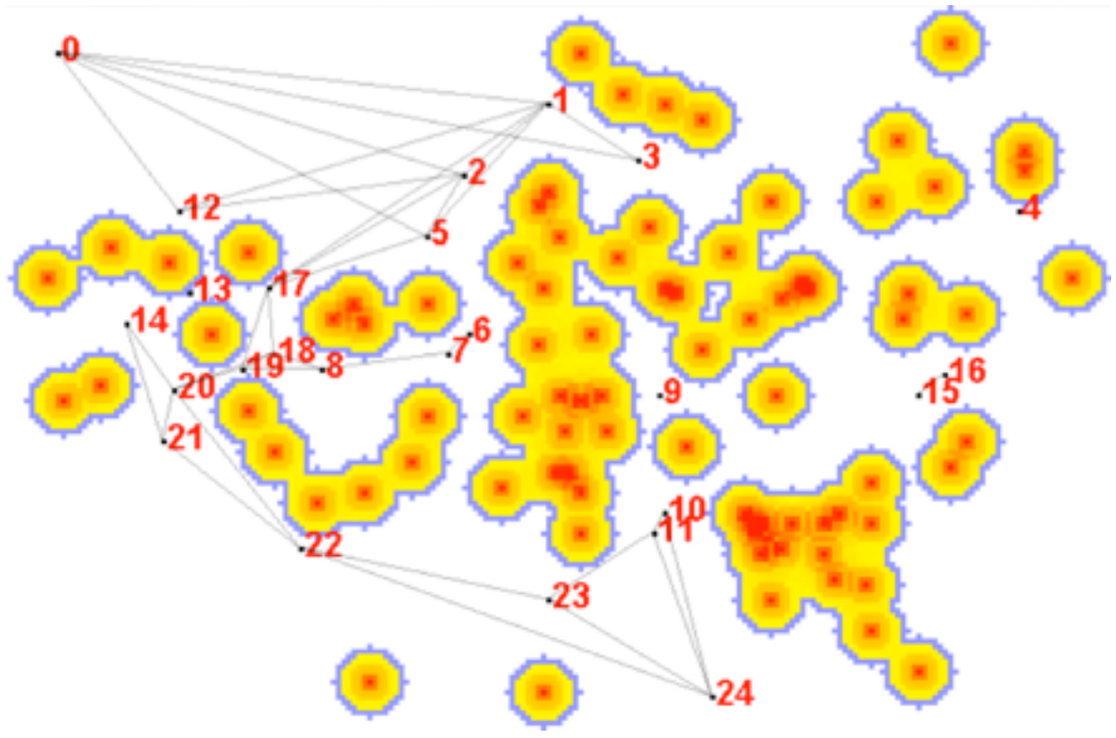
Positionnement : implémentation (2)

```
// détection de la première balise
detecter_balise();
dy=d_prime;

virer(30);  // s'éloigner de la première balise

// détection de la seconde balise
detecter_balise();
dx=d_prime;

if (theta>150) {  // si angle entre les balise > 150 inverser les balises
    swap_dxdy();
    alpha=330-theta;
    dir=270;
} else {
    alpha=theta+30;
    dir=180;
};

// calcul des coordonnées à partir des mesures
```

Extension

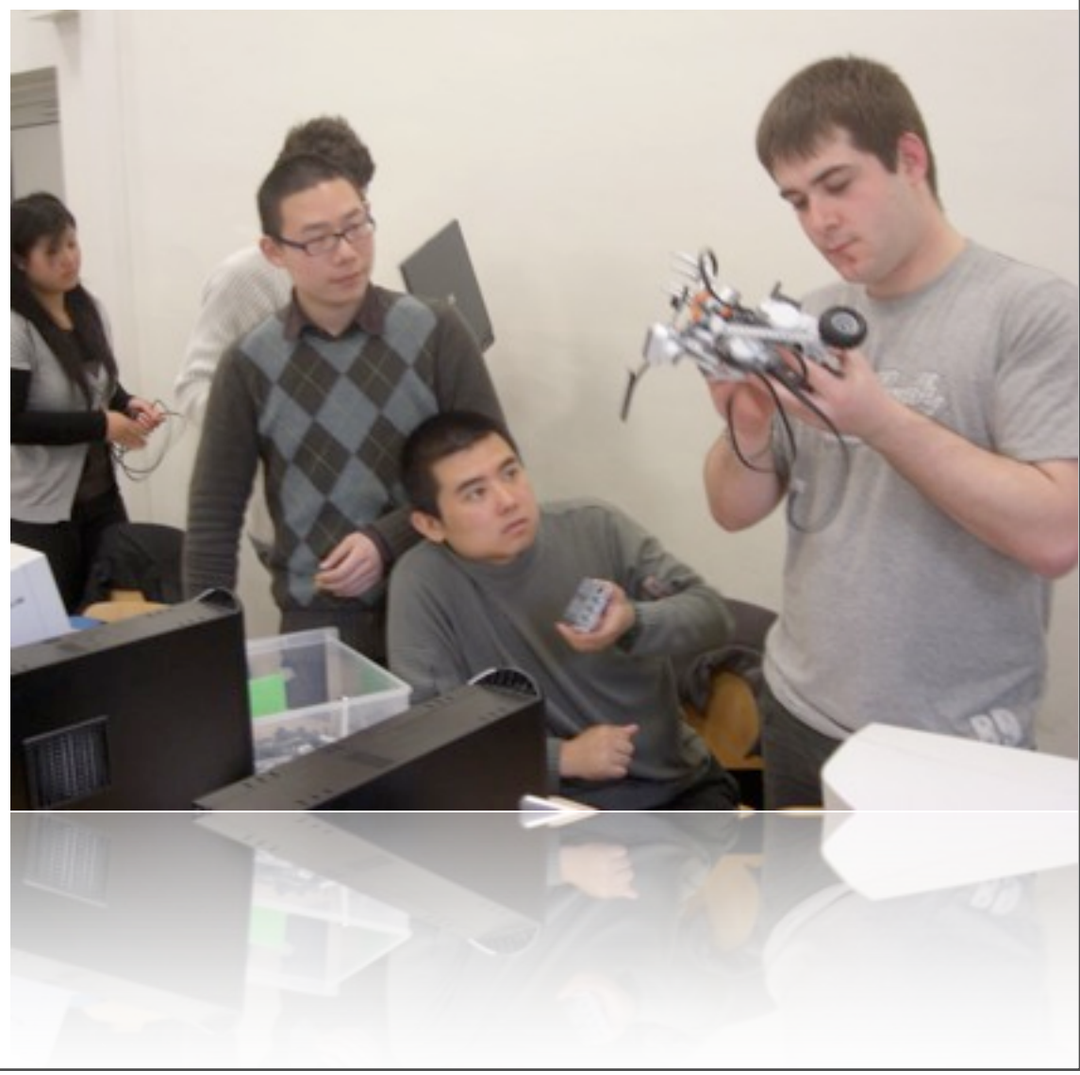
- **Navigation :** rejoindre une position donnée
- **Localisation :** connaître la position courante
- **Cartographie :** dresser la carte de la zone visitée
- **Planification de mission :** décider où aller





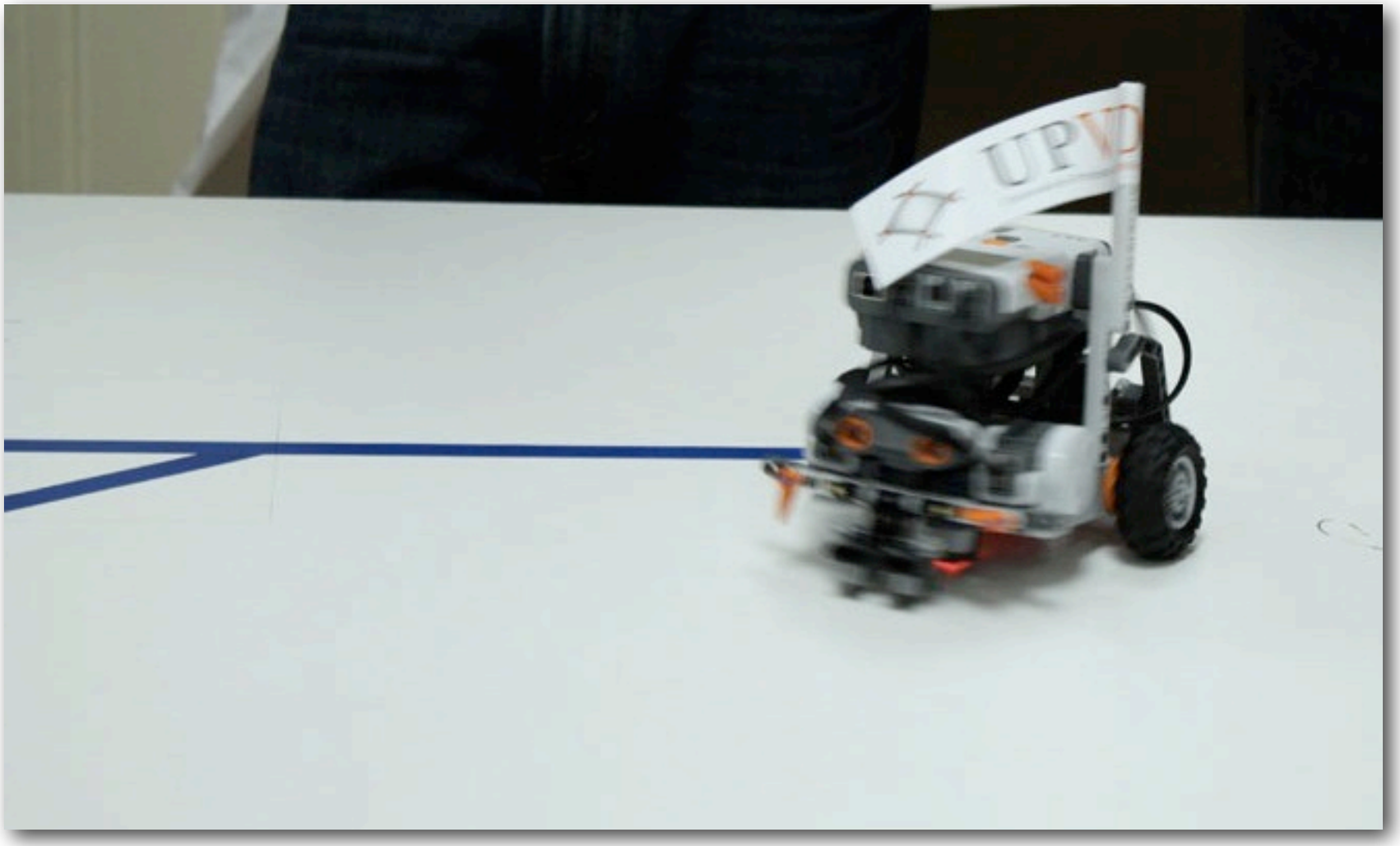
Plan

- Introduction
- Navigation et localisation
- **Autres problèmes**
- Conclusion



Suivi de ligne

Principe : suivre une ligne tracée sur le sol à l'aide du capteur de luminosité



Suivi de ligne : contrôleur PID

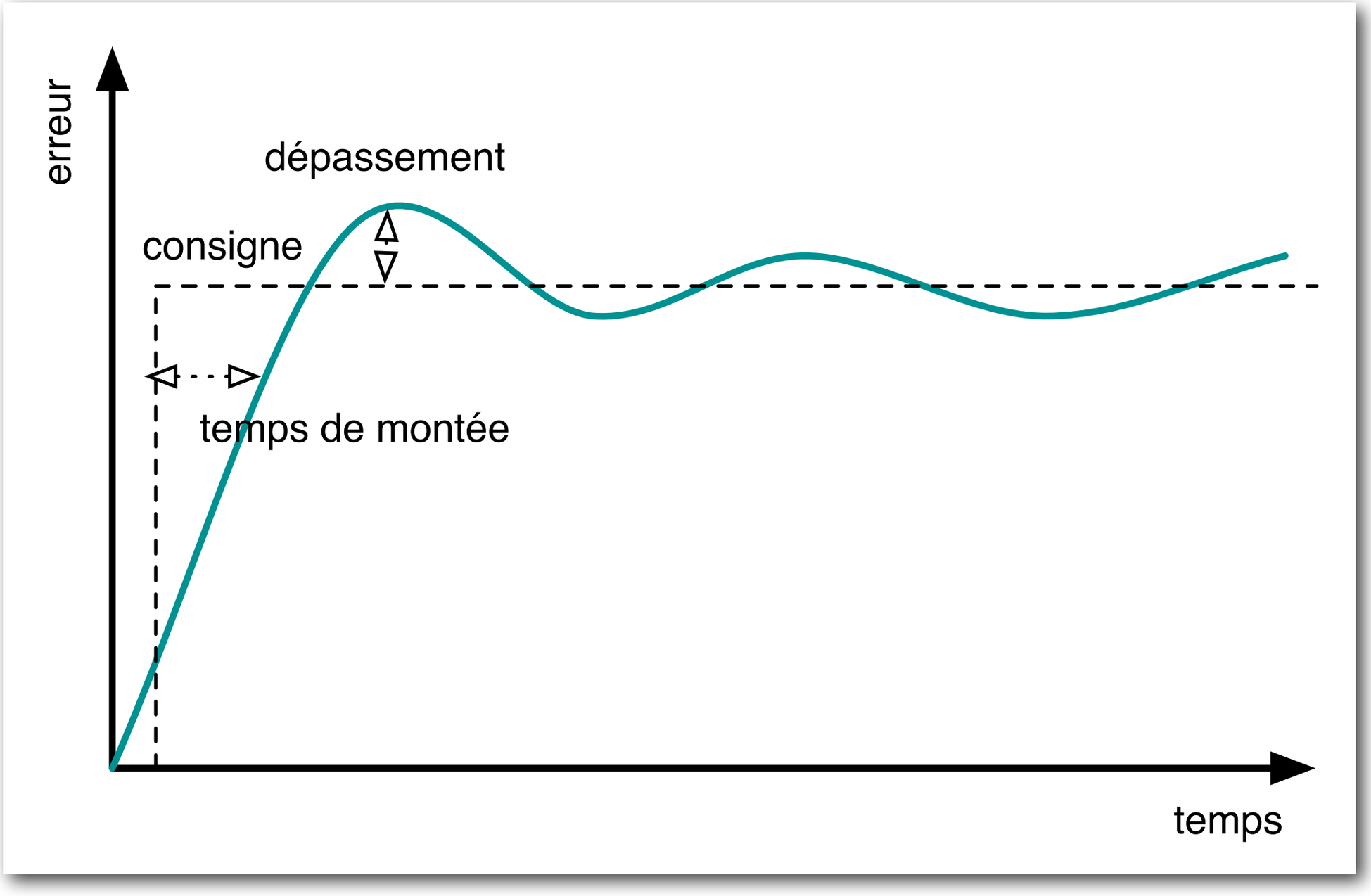
- Principe : maintenir un paramètre à un certain seuil (la consigne)
- Erreur *e(t)* : valeur courante du paramètre – consigne
- Réponse : $$R = K_p \cdot e(t) + K_i \cdot i + K_d \cdot d$$
- *i* terme intégral, *d* terme dérivatif

$$d = \frac{d\ e(t)}{dt}$$

$$i = \int_{\tau=0}^{\tau=t} e(\tau) d\tau$$

- K_p, K_i et K_d constantes appelées gains

PID : comportement usuel



Influence des gains K_p, K_i et K_d sur le temps de réponse, dépassement, oscillations

Implémentation

```
while(SensorUS(IN_4)>15) {          // tant qu'il n'y a pas d'obstacle

    e = SEUIL-Sensor(IN_1);         //   e = erreur courante
    i=i+DT*e;                       //   calcul du terme intégral
    d=(e-e_0)/DT;                   //   calcul du terme dérivatif
    r=KP*e+KI*i+KD*d ;              //   calcul de la réponse

    OnFwd(OUT_B,40+r);              //   application de la réponse aux moteurs
    OnFwd(OUT_C,40-r);

    e_0=e;                          //   màj erreur
    Wait(DT*1000);                  //   attendre DT en ms
}
```

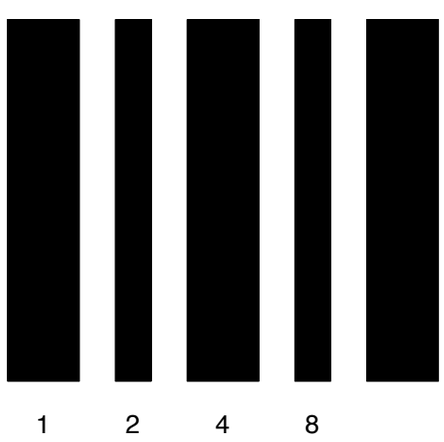
Autre application du PID : équilibre dynamique

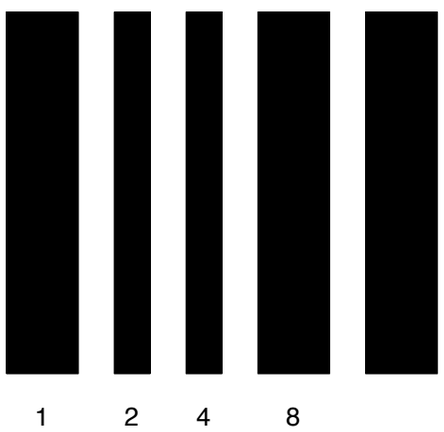
Lecture d'un code barre simple

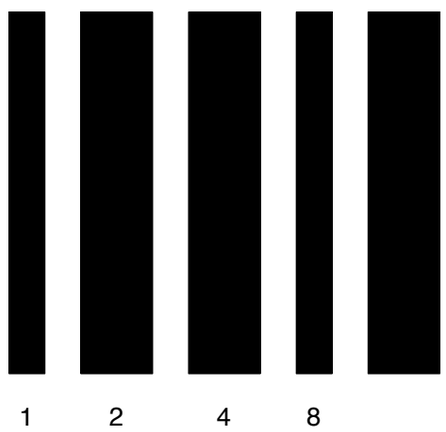
Principe :

$$r = \sum_{i=0}^{n-1} 2^i \cdot b_i$$

- Valeurs données par les barres noires
- Code simple : barre large = 1, barre étroite = 0
- Le code commence systématiquement par une bande large
  - Sert à étalonner
  - Non comptée dans le résultat







Implémentation

```
inline int lire_barre(int a_l) {
    int a;                          // angle de rotation pour traverser la barre
    int res;                        // résultat (0 ou 1)

    a=mesurer_barre();              // a=angle barre courante
    if (a<0.75*a_l) {               // si a < 3/4*a_l alors la barre est fine
        res=0;                          //   le résultat est 0
        Wait(DT);                       //   attendre pour s'éloigner de la barre
    } else {                        // sinon
        res=1;                          //   le résultat est 1
        Wait(DT);                       //   attendre pour s'éloigner de la barre
    };
    return res;
}
```

```
trouver_barre();                // trouver la barre de référence
a_ref=mesurer_barre();          // mesurer l'angle pour parcourir la barre de ref
for(i=0;i<TAILLE_CODE;i++) {    // pour chaque barre du code
    trouver_barre();                                // trouver la barre suivante
    OnFwdReg(OUT_BC,PUISS,OUT_REGMODE_SYNC);        // allumer les moteurs
    Wait(DT);                                       // attendre pour s'éloigner
    b=lire_barre(a_ref);                            // calculer la valeur de la barre
    res=2*res+b;                                    // calcul de la valeur du code
};
```

Exemple de code : le code 39

- Caractère : 5 barres + 4 espaces
- Si barre ou espace large bit = 1, sinon bit = 0
- Exactement 3 éléments larges parmi les 9
- Exemples :
  - A : 100001001
  - B : 001001001
  - C : 101001000

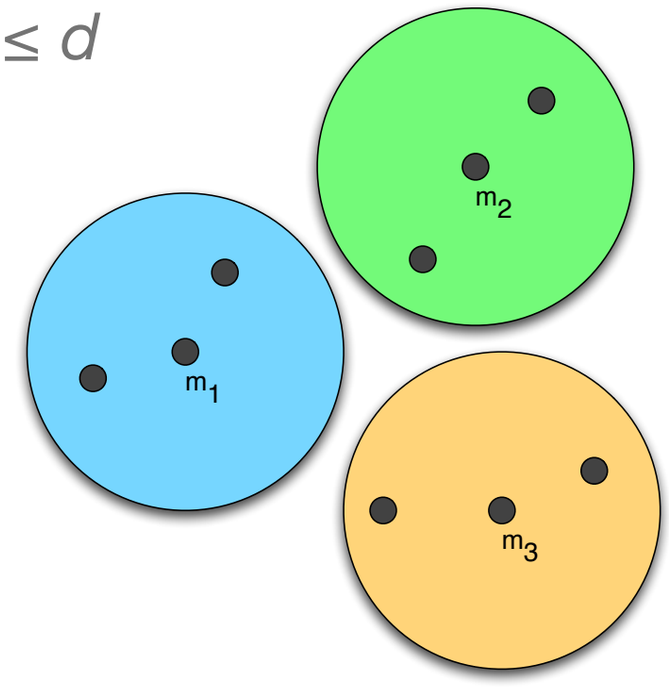
Extension

- Codes correcteurs d'erreurs
- Exemple : Hamming (7,4)
- Distance de Hamming entre deux mots :

$$d(u_0 \dots u_n, v_0 \dots v_n) = Card(\{i,\ 0 \leq i \leq n\ :\ u_i \neq v_i\})$$

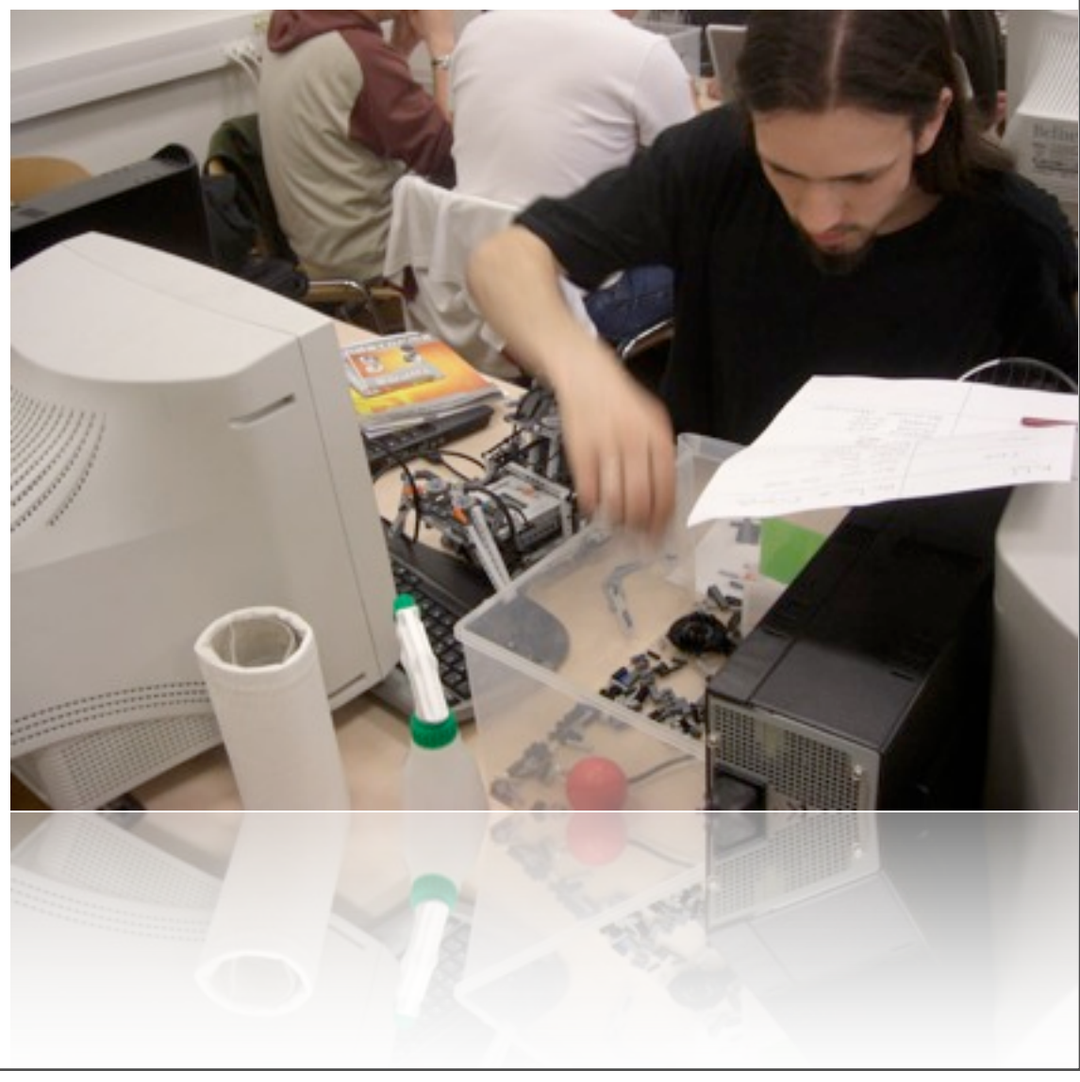
- Soit $M=\{m_1,\dots,m_k\}$ un ensemble de mots distants 2 à 2 d'au moins $d>1$
- m_i, $1 \leq i \leq k$, représente tous les mots m tels que $d(m_i,m) \leq d$

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} = (y_1\ y_2\ \dots\ y_7)$$



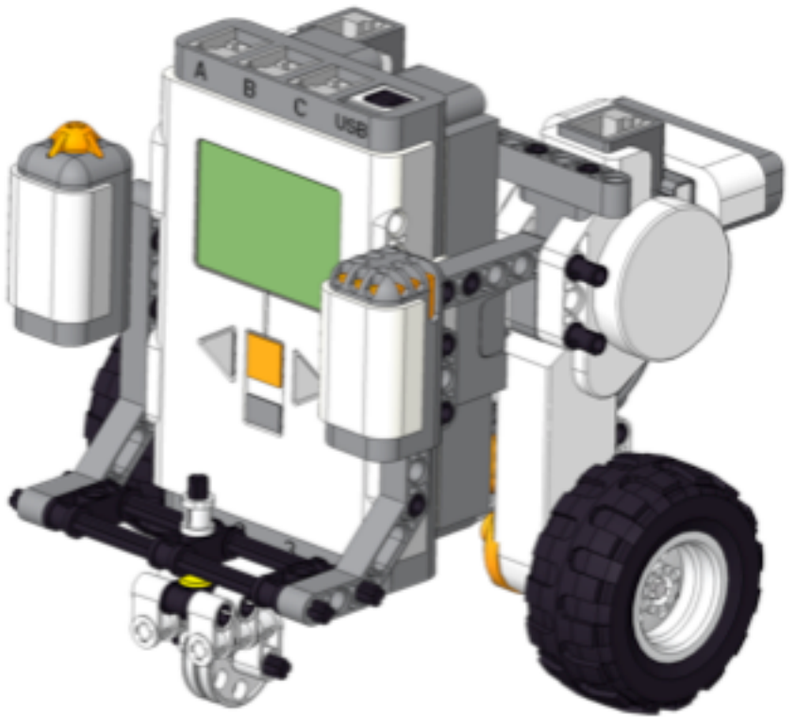
Plan

- Introduction
- Navigation et localisation
- Autres problèmes
- **Conclusion**



Conclusion

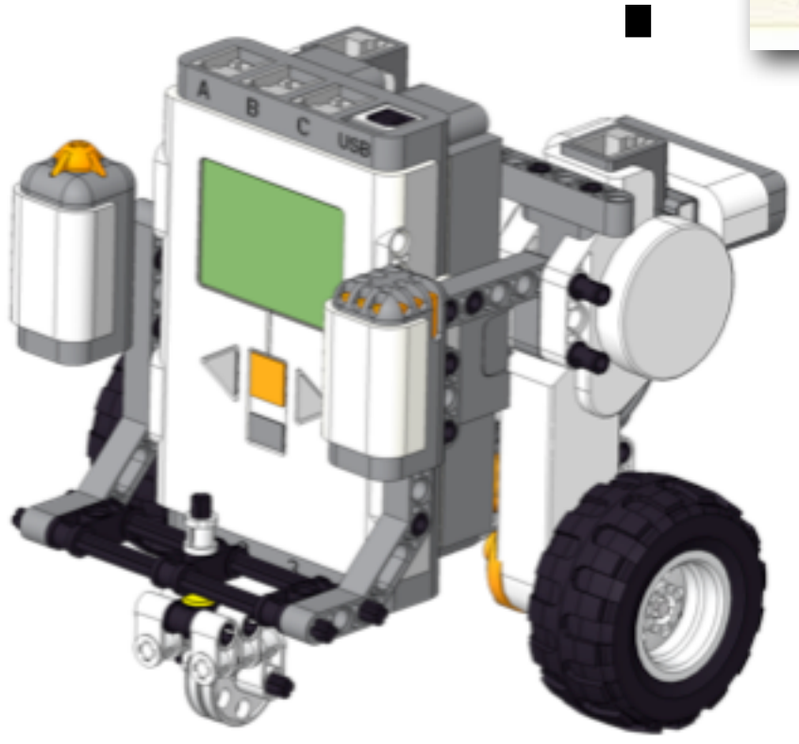
- Trigonométrie
- Courbes paramétrées
- Probabilités
- Intégration et dérivation
- Arithmétique



Conclusion

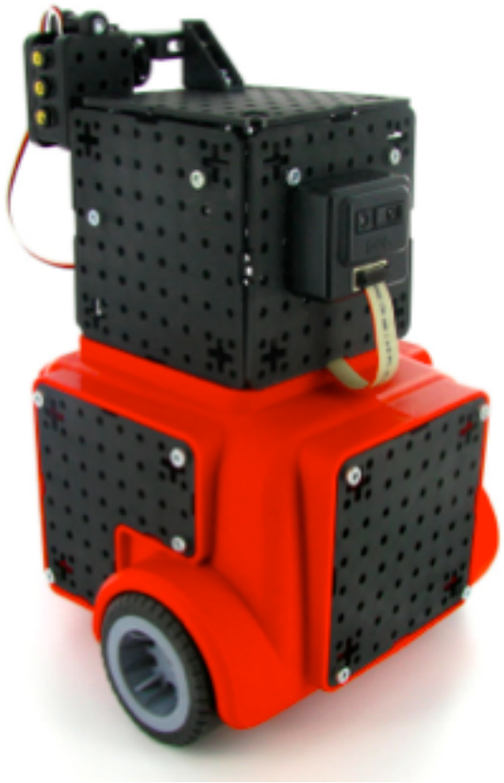
- Trigonométrie
- Courbes paramétrées
- Probabilités
- Intégration et dérivation
- Arithmétique

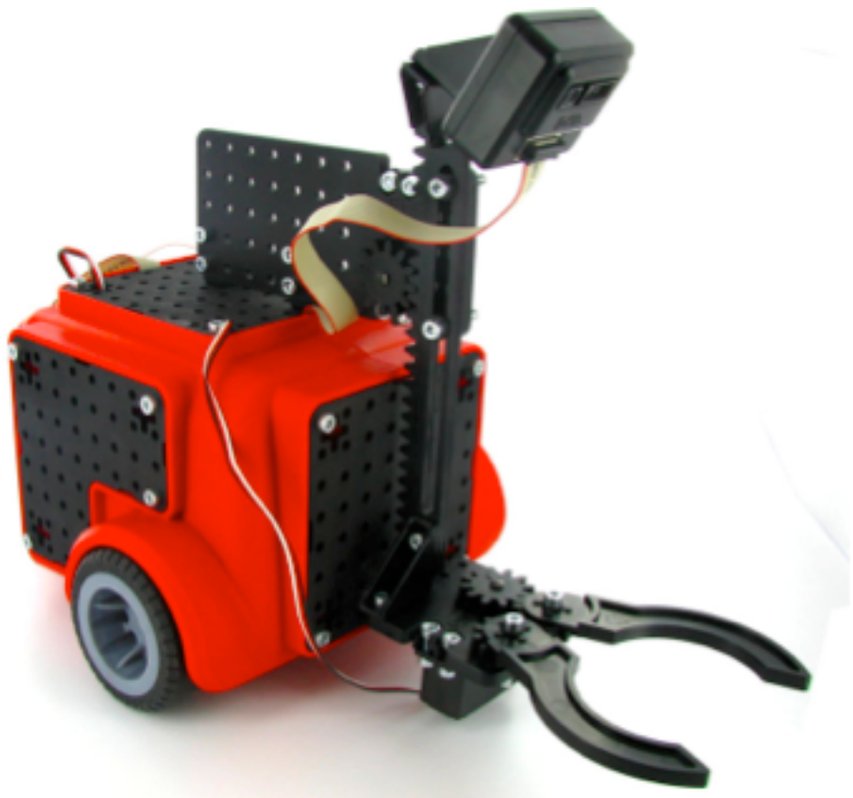
?



Autres kits

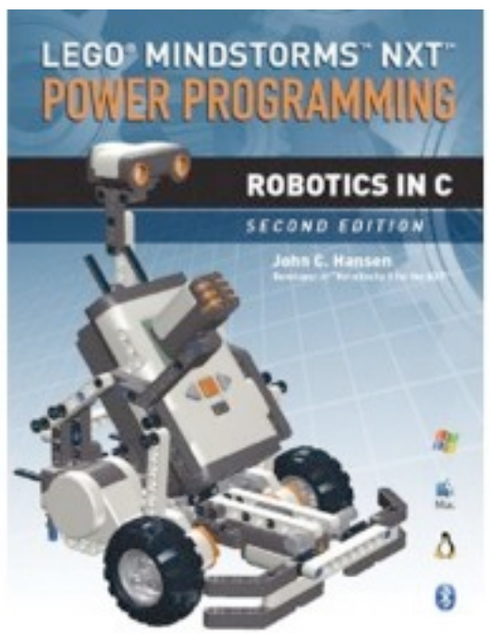
http://www.pob-technology.com/

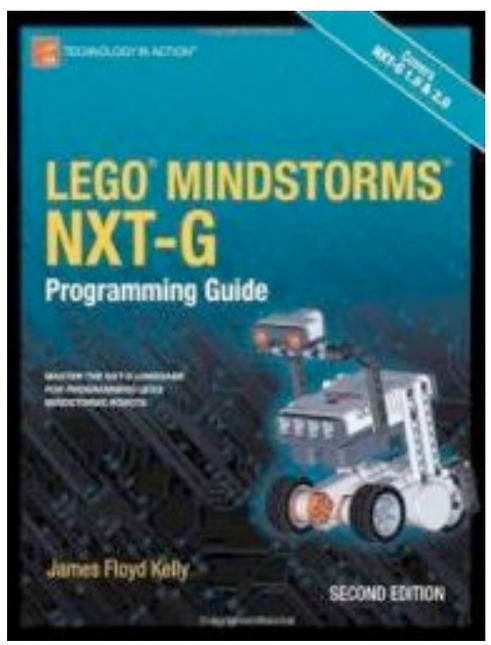




Ressources

- http://mindstorms.lego.com site officiel
- http://www.nxtprograms.com nombreux robots et idées de projets
- http://bricxcc.sourceforge.net/nbc compilateurs NXC et NBC
- http://lejos.sourceforge.net compilateur Java (Lejos)
- Nombreux livres :





Questions?