ISN
Informatique et Sciences du Numérique

Académie d'Aix-Marseille - Formation ISN

INITIATION A LA ROBOTIQUE

MINI-PROJET

Programme de formation :

- Qu'est-ce qu'un robot ?
- A quoi servent les robots ?
- Les kits robotiques
- Les simulateurs
- Application didactique

Les compétences suivantes doivent être développées :

- Identifier les différents composants d'un minirobot et comprendre leurs rôles respectifs.
- Décrire un système à événements simple à l'aide d'une machine à états finis.
- Programmer (dans un langage de haut niveau) un mini robot pour lui faire exécuter une tâche complexe.

Marc Silanus – Lycée A. Benoit – 84800 L'Isle sur Sorgue

Sommaire

1 Définition de la problématique

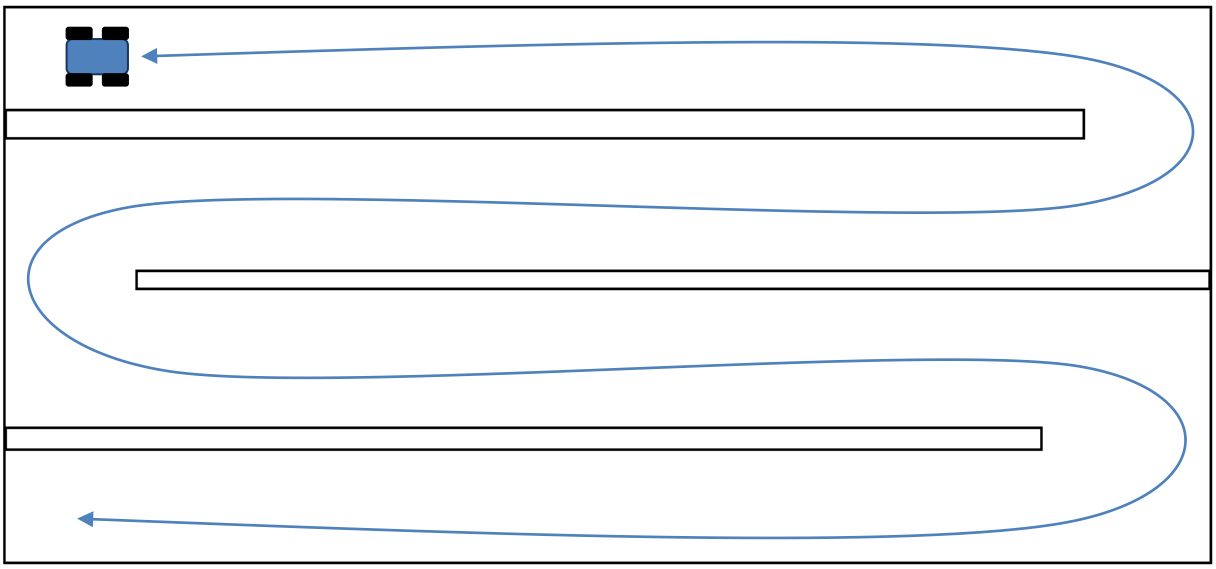
Un hôpital souhaite s'équiper d'un dispositif de transport automatique pour :

- les plateaux repas,
- le linge de lit,
- les traitements médicamenteux, ….

2 Définition des limites du prototype

- Aucune modification des locaux ne peut être envisagée.
- Pas d'émission d'ondes radio
- Déplacement dans les couloirs uniquement

Le prototype devra pouvoir se déplacer dans un circuit tel que celui-ci-dessous :



3 Recherche de solutions

Il s'agit de rechercher une solution matérielle et logicielle qui répond au mieux au cahier des charges.

On se contentera ici d'identifier les éléments minimaux constitutifs d'un prototype pour la validation de l'algorithme de déplacement :

- Un châssis
- Des capteurs
- Un système de traitement de l'information
- Un langage de programmation
- Un comportement

Les quatre premiers points peuvent être imposés par le matériel dont on dispose dans l'établissement.

Pour la formation, on dispose de base robotique Kjunior et Lego Mindstorm, plus éventuellement, des bases robotiques de l'établissement centre de formation.

La recherche de solutions se limitera donc à rechercher comment exploiter au mieux le matériel disponible et imaginer un comportement pour satisfaire à la problématique posée.

4 Apport de connaissances élémentaires

Un apprentissage de l'utilisation du matériel et des logiciels est nécessaire en préalable à l'activité de mini-projet. Cette phase sera aussi l'occasion d'apporter les éléments algorithmiques de base pour réaliser des comportements simples.

Il peut être intéressant d'utiliser un simulateur pour vérifier la cohérence des algorithmes. De plus, le recours à un simulateur permet de palier au manque de matériel et de rendre tous les élèves acteurs de la formation.

L'apport de connaissances élémentaires doit permettre de maitriser les mouvements primitifs su robot :

- Avancer
- Tourner autours d'un point
- Avancer-reculer entre deux obstacles
- Avancer jusqu'à un obstacle, faire un demi-tour et repartir en marche avant
- Suivre un mur (le mur de droite par exemple)

4.1 Utilisation du simulateur Azolla

http://www.codeproject.com/Articles/33587/2D-LUA-Based-Robot-Simulator

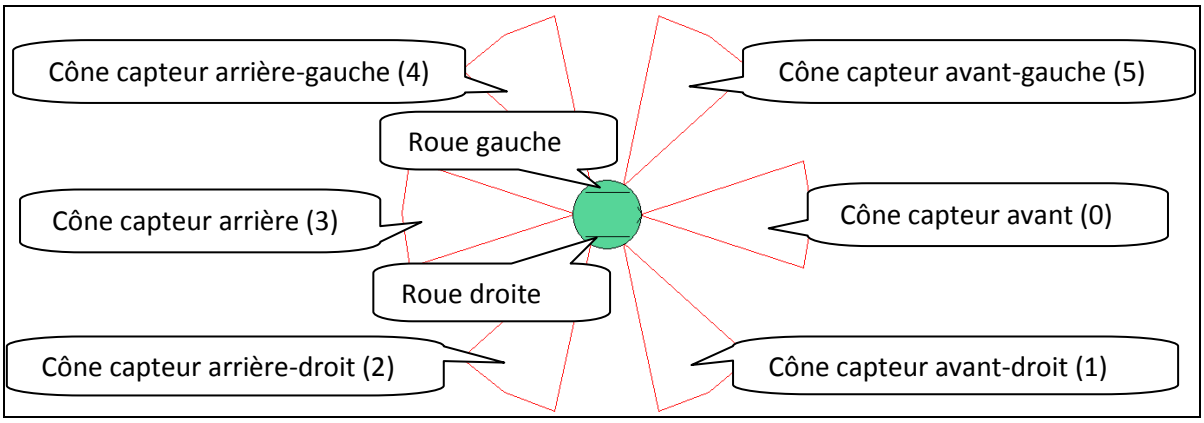
4.1.1 Introduction

Azolla est un simulateur 2D de robot mobile. Avec ce simulateur, nous pouvons concevoir les règles de navigation d'un ou plusieurs robots dans un monde en 2D.

Les règles sont conçues à l'aide de script Lua. C'est un langage de script intégrable puissant, rapide et léger. Utiliser Lua nous offrira de nombreux avantages dans la conception d'algorithmes pour les robots mobiles.

L'éditeur de monde est graphique et intuitif. Les utilisateurs peuvent créer un environnement pour tester les robots en utilisant la souris.

4.1.2 Le robot



4.1.3 Les méthodes d'Azolla

- **`readsensor(integer index)`** : accepts the sensor index; returns the measured distance of the active robot.
- **`setspeed(integer left, integer right)`** : accepts the left and right wheel speed of the active robot; returns nothing.
- **`getangle()`** : accepts nothing; returns the current angular position of the active robot (in radians).
- **`getnumofrobots()`** : accepts nothing; returns number of existing robots.
- **`getposition()`** : accepts nothing; returns x and y position of active robot.
- **`gettarget(int index)`** : accepts index of target; returns x and y position of selected target.
- **`textline(string msg)`** : accepts the message to be displayed; returns nothing.
- **`setactiverobot(integer index)`** : activates a certain robot.
- **`stepforward()`** : runs simulation one time step.

4.1.4 Structure d'un programme

```
function azolla.main(azolla)
    while true do
        -- User code here
    end
end
```

Exemple :

```
function azolla.main(azolla)
    while true do
        -- Read sensors states
        front = azolla:readsensor(0)
        left = azolla:readsensor(1)
        rigth = azolla:readsensor(5)

        -- if there is an obstacle, turn, else forward
        if(front<10 or left<10 or rigth<10) then
                azolla:setspeed(-20,20)
        else
                azolla:setspeed(20,20)
        end
        azolla:stepforward()
    end
end
```

4.2 Tâches élémentaires

4.2.1 Tâche 1 : Avancer

Le robot doit avancer en ligne droite sans limitation de durée ou de distance.

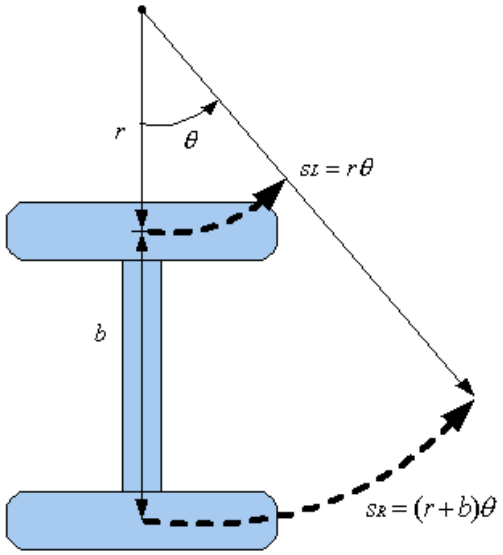
On pourra ensuite imposer au robot de s'arrêter s'il rencontre un obstacle.

4.2.2 Tâche 2 : Tourner autours d'un point

Le robot doit tourner autours d'un point, vers la droite ou vers la gauche. Aucune contrainte angulaire n'est imposée.

On pourra ensuite imposer au robot de tourner :

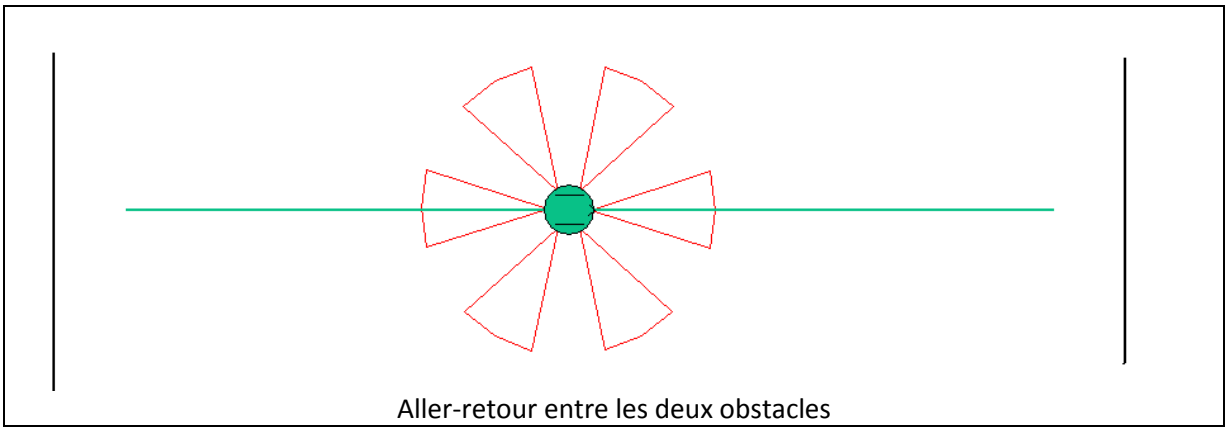
- Sur lui-même,
- Avec un rayon de braquage égal à la distance entre les roues
- Avec un rayon de braquage égal à 2 fois la distance entre les roues.



4.2.3 Tâche 3 : Avancer-Reculer entre deux obstacles

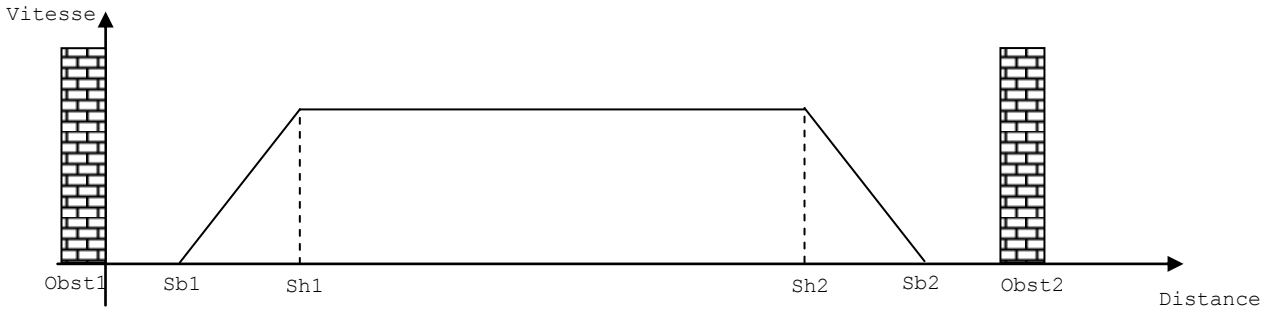
Le robot doit avancer jusqu'à détecter un obstacle devant lui, puis reculer jusqu'à rencontrer un obstacle derrière lui.

Il est nécessaire pour réaliser cette tâche d'utiliser des capteurs de types sonar ultrasons ou infrarouge. On se fixera un seuil de détection minimum correspondant à quelques centimètres avant l'obstacle.



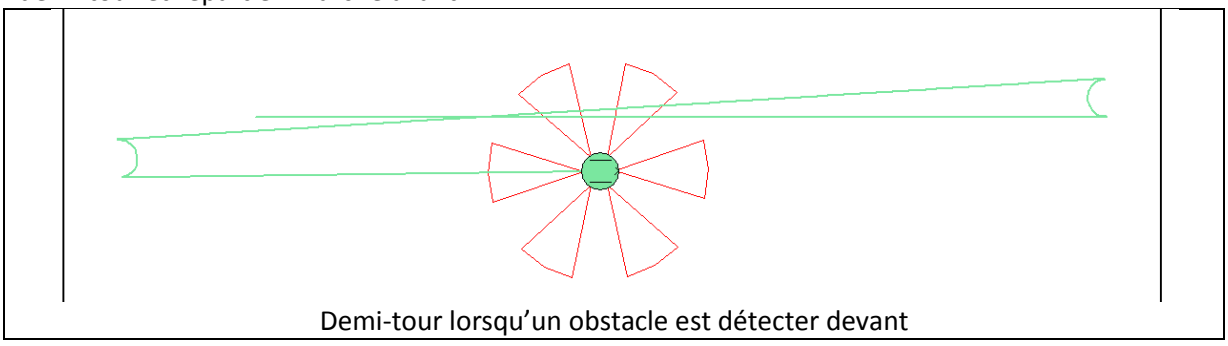
Aller-retour entre les deux obstacles

On pourra ensuite calculer la loi de vitesse à imposer aux roues pour ralentir avant l'obstacle, s'arrêter et repartir en marche arrière.



4.2.4 Tâche 4 : Avancer jusqu'à un obstacle, faire un demi-tour et repartir en marche avant

Le robot doit toujours rouler en marche avant. Lorsqu'il rencontre un obstacle devant lui, il fait demi-tour et repart en marche avant.



Demi-tour lorsqu'un obstacle est détecter devant

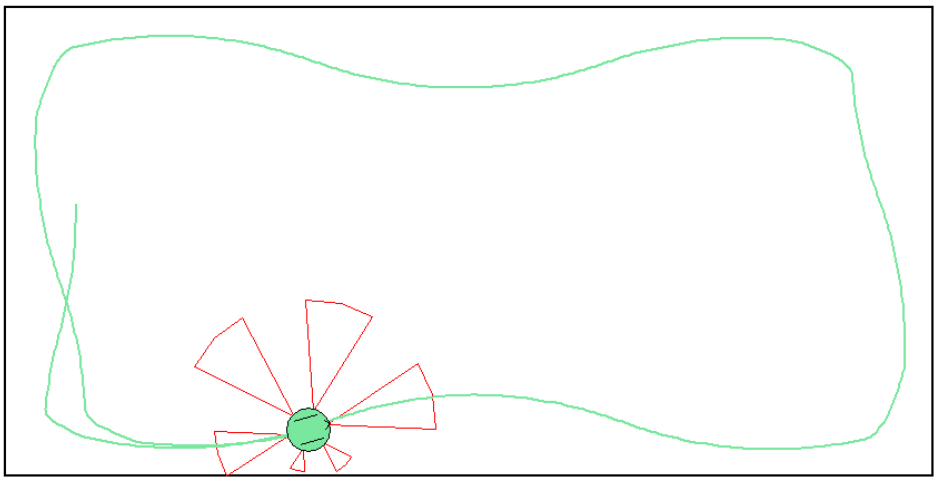
On pourra ensuite placer quelques obstacles dans la zone d'évolution du robot et observer son comportement. On pourra réfléchir aux éléments d'amélioration de la détection d'obstacles (matériel et/ou logiciel) à apporter pour éviter que le robot ne se retrouve bloqué.

Dans quel type d'application cet algorithme pourrait-il être employé ?

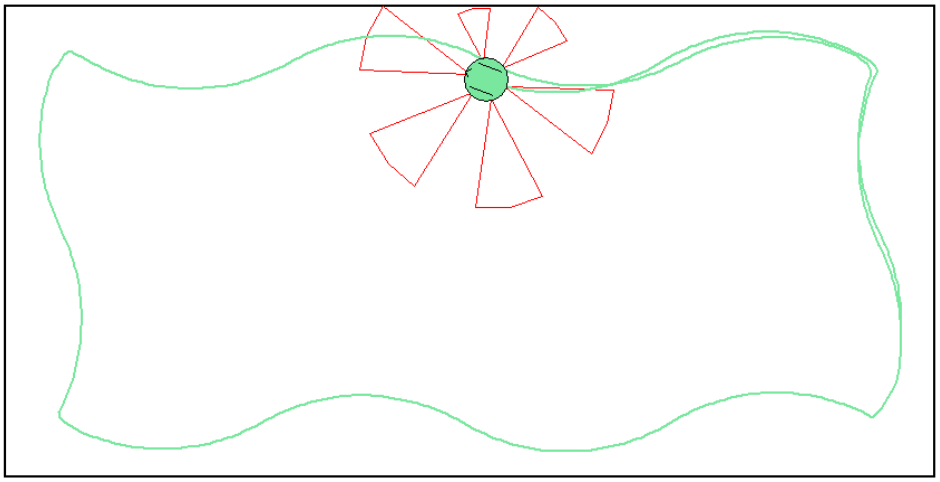
4.2.5 Tâche 5 : Suivre un mur

Le robot doit suivre un mur (celui de droite par exemple). On essaiera de minimiser au maximum l'effet de gigue.

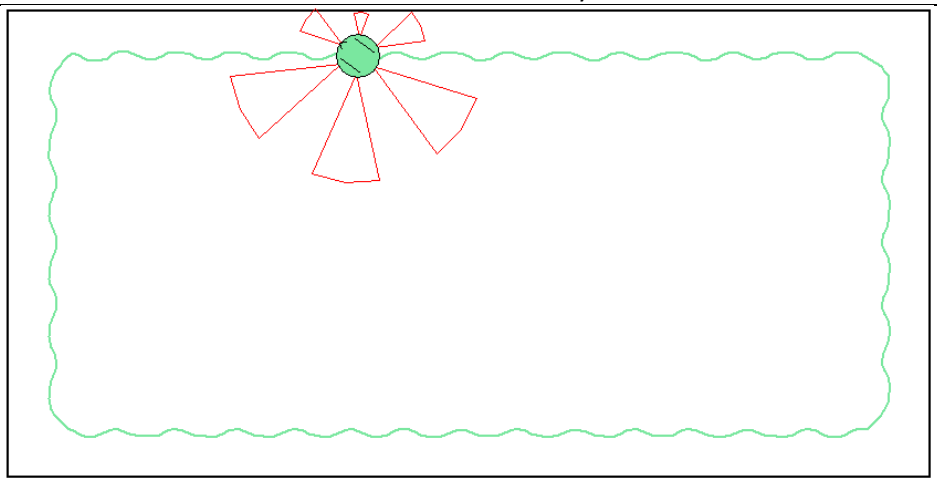
Le robot doit rester à une distance de 5cm du bord du mur +/- 2,5mm.



Vitesse de correction faible



Vitesse de correction moyenne



Vitesse de correction forte

Jouez sur les différents paramètres et observez le comportement du robot.

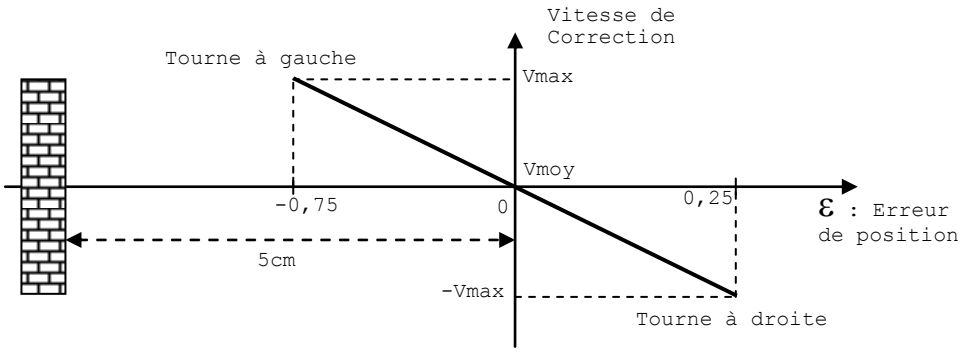
Tentative d'amélioration du comportement.

L'observation du comportement montre que :

- lorsqu'on augmente la vitesse de correction, on obtient une grande instabilité (fréquence de la gigue) mais une faible amplitude de l'erreur de position par rapport au mur. Le robot avance lentement.
- lorsque l'on diminue la vitesse de correction, on obtient une meilleure stabilité mais une plus grande amplitude de l'erreur de position par rapport au mur. Le robot avance plus vite.

La vitesse de correction a donc une influence sur la position par rapport au mur et sur la vitesse globale de déplacement du robot. Dans l'algorithme précédent, on a corrigé la trajectoire en imposant une vitesse de correction identique quelque soit la position du robot par rapport au mur.

- Il serait bon de pouvoir moduler cette vitesse de correction en fonction de la position du robot. Une solution envisageable est de calculer une loi de vitesse permettant d'augmenter cette vitesse de correction lorsque le robot s'éloigne de la position voulue et de la diminuer lorsque le robot se rapproche de la position voulue.



Ce type de correcteur s'appelle « Correcteur proportionnel ».

L'expression de la vitesse de correction est donc : $V_C = \frac{2V_{max}}{0,5}.\varepsilon = 4V_{max}.\varepsilon$

On appliquera donc les vitesses suivantes sur les moteurs :

- Moteur droit : `Vmoy + Vc`
- Moteur gauche : `Vmoy – Vc`

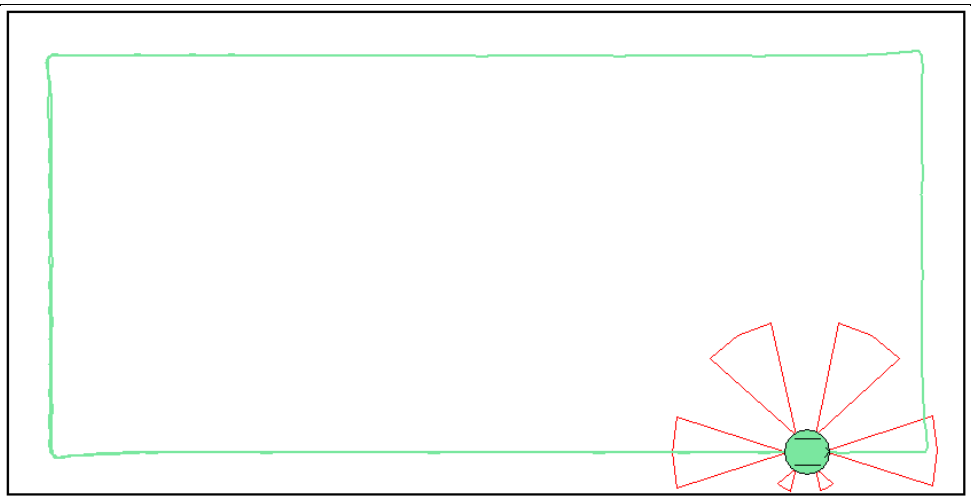
Remarque : Le coefficient de correction est généralement noté Kp (coefficient du correcteur proportionnel). Ici, `Kp=4Vmax`

On pourra modifier la valeur de Kp et vérifier le comportement du robot :

Suivre un tracé rectiligne est d'ordinaire assez facile. Suivre un tracé comportant des courbes douces est un peu plus délicat. Suivre enfin un tracé comportant des courbes serrées est nettement plus dur.

Si le robot se déplace assez lentement, presque n'importe quel tracé peut être suivi.

Pour suivre un tracé, avec une vitesse convenable et la capacité d'aborder avec succès des virages doux, il est nécessaire d'apporter une correction proportionnelle à l'erreur. Cependant, le meilleur correcteur proportionnel est spécifique à chaque type de tracé (largeur, rayons des courbes, éclairage, etc.) et à chaque robot. En d'autres termes, un correcteur P est conçu pour une seule sorte de tracé et de robot et ne fonctionnera pas convenablement pour d'autres tracés ou robots. Il faudra donc adapter les paramètres Kp et Vmoy selon les circonstances.



Avec Kp = 15 et Vmoy = 5

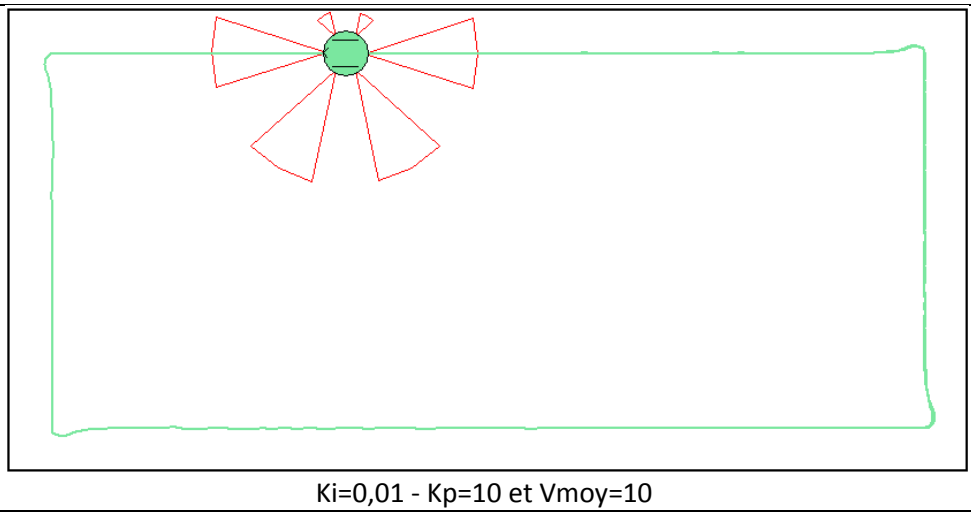
Lorsque l'erreur de position est faible, la correction est faible. Si l'erreur persiste, il faut longtemps au correcteur proportionnel pour agir. On serait tenter d'augmenter la valeur de Kp mais dans se cas, le robot se met à osciller.

L'idée dans se cas est de calculer le cumul des erreurs pendant un certain temps et le multiplier par un coefficient pour ajuster la correction de vitesse :

$$V_c = K_p.\varepsilon + K_I.\Sigma\,(\varepsilon.\Delta t)$$

Ce type de correcteur s'appelle « Correcteur intégrale ».

La somme des erreurs courantes est renouvelée à chaque itération du programme (Δt). Il s'agit donc d'un calcul intégral. Le temps entre 2 itérations étant constant, Δt peut faire partie du coefficient intégral Ki. Ce dernier pourra être ajusté par tâtonnements.



Ki=0,01 - Kp=10 et Vmoy=10

Le facteur intégral est l'histoire cumulative des erreurs et donne au correcteur un moyen de les corriger quand elles persistent pendant une longue période de temps.

La meilleure solution serait sans doute de pouvoir prévoir les erreurs pour essayer de les corriger avant qu'elles se soient produites.

En supposant que la prochaine variation de l'erreur est identique à sa dernière variation, on peut en déduire que :

$$ErreurFuture = ErreurActuelle + derive.\Delta t$$

La dérive de l'erreur est donnée par la pente de la droite d'erreur, c'est-à-dire la dérivée de l'erreur :

$$derive = (ErreurActuelle - ErreurPrécédente)/\Delta t$$

En général, les correcteurs dérivés se contentent de la dérive de l'erreur et ne calculent pas l'erreur suivante :

$$V_c = K_p.\varepsilon + K_I.\sum \varepsilon + K_d \frac{\Delta\varepsilon}{\Delta t}$$

Comme pour le coefficient intégral, le terme Δt étant constant, il peut faire partie du coefficient Kd.

Le correcteur total obtenu est nommé PID : Proportionnel-Intégral-Dérivé

$$V_c = K_p.\varepsilon + K_I.\sum \varepsilon + K_d.\Delta\varepsilon$$

Remarque : Deux méthodes sont particulièrement répandues pour le réglage des paramètres d'un correcteur PID. La première est entièrement théorique, il s'agit de la méthode de « Naslin ». La seconde se base sur l'expérience et est donc beaucoup moins calculatoire, il s'agit de la méthode de « Ziegler et Nichols ».

L'expression du PID est souvent notée dans le domaine de Laplace :

$$C(p) = K_p.\left(1 + \frac{1}{T_I p} + T_d.p\right)$$

Le principe de la méthode de « Ziegler et Nichols » consiste à amener, en augmentant le gain Kp, un système non corrigé à la limite d'oscillation. On mesure la période T0 des oscillations et le gain statique rajouté correspondant (Kp=K0). Ensuite on utilise le tableau ci-dessous (source INPG) :

	Kp	Ti	Td
correcteur P	$0,5\ K_0$	-	-
correcteur PI	$0,45\ K_0$	$0,8\ T_0$	-
correcteur PID	$0,6\ K_0$	$0,5\ T_0$	$0,125\ T_0$

En conclusion :

- Kp contrôle le "niveau" de correction d'erreur et donc, plus Kp est grand, plus la correction est rapide mais plus le dépassement est important : si ma réaction est trop forte je dépasse l'objectif de beaucoup.
- Ki permet de limiter l'erreur, lorsque le système est "stable" (lorsqu'il a cessé d'osciller). Ki contrôle donc la précision : une valeur de Ki élevée augmente la précision mais augmente le temps nécessaire à stabiliser le système.
- Kd permet de limiter le dépassement. Lorsque Kd augmente, on limite l'amplitude des oscillations mais le système en « anticipant » de trop, met plus de temps à atteindre l'objectif.

Pour en savoir plus, reporter vous au cours « Introduction aux systèmes automatisés – Asservissement et régulation »

5 Concevoir un prototype pour répondre à la problématique

A partir des choix matériels établis précédemment, assemblez puis programmez votre robot pour qu'il réponde à notre problématique.

6 Exemple de solutions

6.1 Tâche 1 : Avancer

6.1.1 Azolla

```
function azolla.main(azolla)
    while true do

                front = azolla:readsensor(0)

                if(front<10) then
                        azolla:setspeed(0,0)
                else
                        azolla:setspeed(20,20)
                end
                azolla:stepforward()
    end
end
```

6.1.2 Kjunior

```
#include "KJunior.h"

void main(void)
{
   // Initialization
   KJunior_init();
   KJunior_config_auto_refresh_sensors(MANUAL);

   while(1)
   {
      KJunior_flag_sensors_reset();
      KJunior_manual_refresh_sensors();
      while(Sensors_Refreshed_Flag == 0);  // Wait until the sensor are refreshed

      if(KJunior_get_proximity(FRONT) > 400) KJunior_set_speed(0,0);
      else KJunior_set_speed(9,9);
   }
}
```

6.1.3 RobotC Lego Mindstorm

```
#pragma config(Sensor, S1,     front,          sensorTouch)
//*!!Code automatically generated by 'ROBOTC' configuration wizard !!*//

task main()
{
        while(true)
        {
                if(SensorValue[front]==1)
                {
                        motor[motorA]=0;
                        motor[motorC]=0;
                }
                else
                {
                        motor[motorA]=100;
                        motor[motorC]=100;
                }
        }
}
```

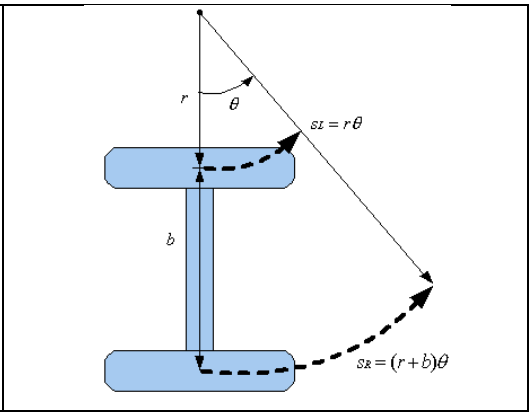
6.2 Tâche 2 : Tourner autours d'un point

- Tourner sur lui-même. Centre de rotation au centre de l'axe des roues : Vitesse de même valeur mais de sens opposé à chaque roue.
- Rayon de braquage égal à la distance entre les roues. Le centre de rotation se situe sur la roue qui sert de pivot : Vitesse nulle à cette roue.
- Rayon de braquage égal à 2x la distance entre les roues : Le centre de rotation se situe à r=b.

⇒ Vitesse roue 1 : $v1 = r.\frac{d\theta}{dt}$

⇒ Vitesse roue 2 : $v2 = 2.r.\frac{d\theta}{dt} = 2.V1$

Attention : On ne tient pas compte des frottements, jeux mécaniques, déformation des axes, …. **Ces vitesses sont purement théoriques**.




```
function azolla.main(azolla)
    while true do
                azolla:setspeed(-10,10) //Tourner sur lui-même
    end
end
```

```
function azolla.main(azolla)
    while true do
                azolla:setspeed(0,10) //rayon de braquage = distance entre les roues
    end
end
```

```
function azolla.main(azolla)
    while true do
                azolla:setspeed(5,10) //rayon de braquage = 2x distance entre les roues
    end
end
```

6.2.1 Kjunior

```
#include "KJunior.h"
void main(void)
{
       KJunior_init()
       while(1)
       {
            KJunior_set_speed(-9,9); //Sur lui même
       }
}
```

```
while(1)
{
       KJunior_set_speed(0,9); //rayon braquage = distance entre les roues
}
```

```
while(1)
{
        KJunior_set_speed(4,8); // rayon braquage = 2x distance entre les roues
}
```

6.2.2 RobotC Lego Mindstorm

```
task main()
{
      while(true)
      {
            {
               motor[motorA]=100;
               motor[motorC]=-100; //Sur lui-même
            }
      }
}
```

```
{
      motor[motorA]=100;
      motor[motorC]=0; // rayon braquage = distance entre les roues
}
```

```
{
      motor[motorA]=100;
      motor[motorC]=50; // rayon braquage = 2x distance entre les roues
}
```

6.3 Tâche 3 : Avancer-Reculer entre deux obstacles

6.3.1 Azolla

```
function azolla.main(azolla)
    while true do
                front = azolla:readsensor(0)
                back = azolla:readsensor(3)
                -- Définir le sens
                if(front<12) then sens=-1
                end
                if(back<12) then sens=1
                end
                -- Vitesse constante
                if(front>20 and back>20) then vitesse=20;
                end
                -- Accéléré ou ralentir
                if(front<20) then
                       vitesse=2*front-20;
                end
                if(back<20) then
                       vitesse=2*back-20;
                end
                azolla:setspeed(sens*vitesse+1,sens*vitesse+1)
                azolla:stepforward()
    end
end
```

6.3.2 Kjunior

```
#include "KJunior.h"
int sens=1;
int vitesse=10;
void main(void)
{
   // Initialization
   KJunior_init();
   KJunior_config_auto_refresh_sensors(MANUAL);
   while(1)
   {
      KJunior_flag_sensors_reset();
      KJunior_manual_refresh_sensors();
      while(Sensors_Refreshed_Flag == 0);  // Wait until the sensor are refreshed
      KJunior_set_speed(vitesse*sens,vitesse*sens);
      if(KJunior_get_proximity(FRONT) > 400) sens=-1;
      if(KJunior_get_proximity(REAR) > 400) sens=1;
      if(KJunior_get_proximity(FRONT) < 200 && KJunior_get_proximity(REAR) <200)
        vitesse=10;
      if(KJunior_get_proximity(FRONT) >200)
        vitesse=(int8)(-0.02* KJunior_get_proximity(FRONT)+14);
      if(KJunior_get_proximity(REAR) >200)
        vitesse=(int8)(-0.02* KJunior_get_proximity(REAR)+14);
   }
}
```

6.3.3 RobotC Lego Mindstorm

```
#pragma config(Sensor, S1,     front,          sensorSONAR)
#pragma config(Sensor, S2,     back,           sensorSONAR)
//*!!Code automatically generated by 'ROBOTC' configuration wizard  !!*//
task main()
{
     int sens=1;
     int vitesse=0;
     while(true)
     {
        if(SensorValue[front]<12) sens=-1;
        if(SensorValue[back]<12) sens=1;
        if(SensorValue[front]>110 && SensorValue[back]>110) vitesse=100;
        if(SensorValue[front]>10 && SensorValue[front]<110) vitesse=SensorValue[front]-10;
       if(SensorValue[back]>10 && SensorValue[back]<110) vitesse=SensorValue[back]-10;
       motor[motorA]=sens*vitesse;
       motor[motorC]=sens*vitesse;
     }
}
```

6.4 Tâche 4 : Avancer jusqu'à un obstacle, faire un demi-tour et repartir en marche avant

6.4.1 Azolla

```
function azolla.main(azolla)
    azolla:setspeed(10,10)
    while true do

        front = azolla:readsensor(0)
        back = azolla:readsensor(3)

        if(front<2) then
               azolla:setspeed(0,-9)
        end
        if(back<3) then
               azolla:setspeed(10,10)
        end
        azolla:stepforward()
    end
end
```

6.4.2 Kjunior

```
#include "KJunior.h"
void main(void)
{
   // Initialization
   KJunior_init();
   KJunior_config_auto_refresh_sensors(MANUAL);

   while(1)
   {
      KJunior_flag_sensors_reset();
      KJunior_manual_refresh_sensors();
      while(Sensors_Refreshed_Flag == 0);    // Wait until the sensor are refreshed
      if(KJunior_get_proximity(FRONT) > 400) KJunior_set_speed(-9,0);
      if(KJunior_get_proximity(REAR) > 400) KJunior_set_speed(9,9);
   }
}
```

6.4.3 RobotC Lego Mindstorm

```
#pragma config(Sensor, S1,     front,          sensorSONAR)
#pragma config(Sensor, S2,     back,           sensorSONAR)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

task main()
{
       motor[motorA]=50;
       motor[motorC]=50;
       while(true)
       {
              if(SensorValue[front]<10)
              {
                     motor[motorA]=-50;
                     motor[motorC]=0;
              }
              if(SensorValue[back]<10)
              {
                     motor[motorA]=50;
                     motor[motorC]=50;
              }
       }
}
```

6.5 Tâche 5 : Suivre un mur

6.5.1 Azolla

```
function azolla.main(azolla)
	while true do
		front = azolla:readsensor(0)
		right = azolla:readsensor(1)

		if(front<10) then azolla:setspeed(-10,10)
		else
			if(right>5.25) then azolla:setspeed(4,5)	--Correction
			else if(right<4.75) then azolla:setspeed(5,4) --Correction
				else azolla:setspeed(5,5)
				end
			end
		end
		azolla:stepforward()
	end
end
```

6.5.2 Kjunior

```
#include "KJunior.h"
void main(void)
{
   KJunior_init();
   KJunior_config_auto_refresh_sensors(MANUAL);
   KJunior_set_speed(9,9);
   while(1)
   {
      KJunior_flag_sensors_reset();
      KJunior_manual_refresh_sensors();
      while(Sensors_Refreshed_Flag == 0);  // Wait until the sensor are refreshed
      if(KJunior_get_proximity(FRONT) > 400) KJunior_set_speed(0,0);
      else if(KJunior_get_proximity(RIGHT) > 500) KJunior_set_speed(0,9); //correction
      else if(KJunior_get_proximity(RIGHT) < 400) KJunior_set_speed(9,0); //correction
      else KJunior_set_speed(9,9);
   }
}
```

6.5.3 RobotC Lego Mindstorm

```
#pragma config(Sensor, S1,     front,          sensorSONAR)
#pragma config(Sensor, S2,     right,          sensorSONAR)

task main()
{
	motor[motorA]=50;
	motor[motorC]=50;
	while(true)
	{
		if(SensorValue[front]<10)
		{
			motor[motorA]= 0;
			motor[motorC]= 0;
		}
		if(SensorValue[right]<20)
		{
			motor[motorA]=50;
			motor[motorC]=40;
		}
		else
		if(SensorValue[right]>22)
		{
			motor[motorA]=40;
			motor[motorC]=50;
		}
		else
		{
			motor[motorA]=50;
			motor[motorC]=50;
		}
	}
}
```

6.6 Tâche 5 : Suivre un mur – Tentative d'amélioration

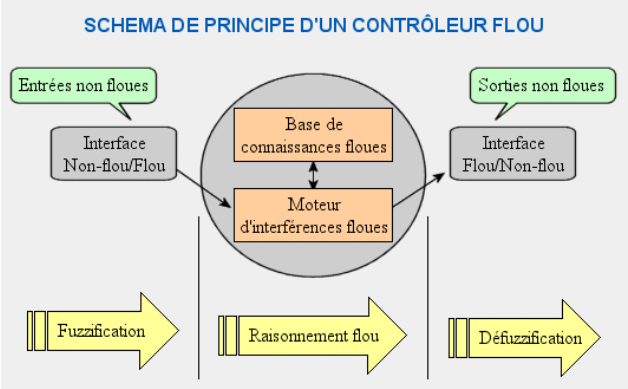
Plusieurs solutions sont possibles. La correction PID étant sans doute la meilleurs, mais elle n'est pas toujours applicable. Ainsi, le robot K-junior ne disposant que de 9 vitesses entières et seulement 6 ou 7 effectives (de 0 à 3 quasiment inutilisable à cause des frottements et de la mécanique), se prête très mal à la programmation d'un tel correcteur. On peut en revanche appliquer un raisonnement de logique floue :

6.6.1 La logique floue

- La Fuzzification consiste à définir les ensembles, les variables linguistiques et définir leurs fonctions d'appartenance. C'est l'opération qui transforme les valeurs réelles d'une variable en quantité floue.
- Le raisonnement floue s'appui sur une base de connaissances constituée de règles du type :

  Si (X est A) **Alors** (Y est B)
  - ✓ X et Y sont des variables linguistiques
  - ✓ A et B sont des classes floues

  Le moteur d'inférences applique les règles sur les entrées pour fournir les sorties.
- La défuzzification est l'opération qui transforme une quantité floue en une valeur réelle. On utilise en général la méthode du centre de gravité (COG).

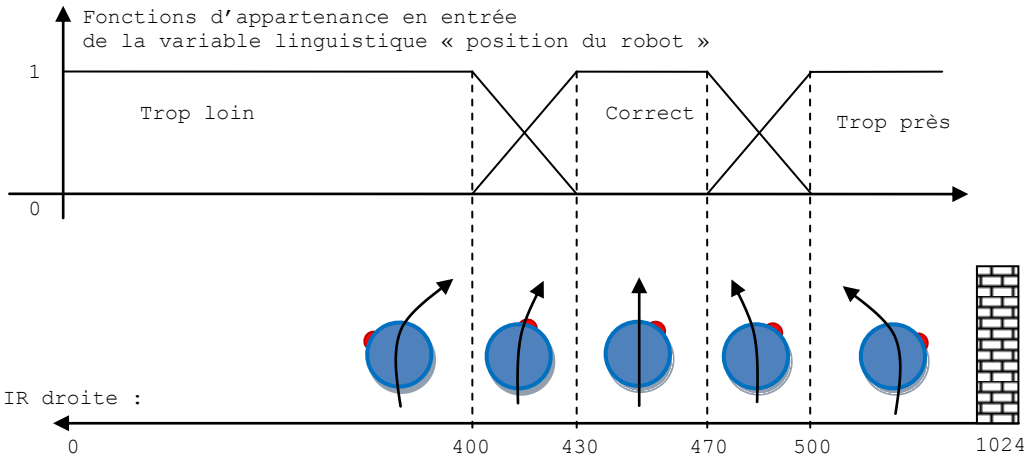


6.6.2 Fuzzification

Le capteur IR droit nous informe sur la position du robot par rapport au mur à suivre. Il nous fournit un nombre compris entre 0 et 1023. Le mur est très loin pour 0 et au contact pour 1023. La position idéale du robot se situe autours de la valeur 450. C'est l'expérience qui permet de trouver cette valeur.

Autours de cette valeur, on accepte une dérive entre 400 et 500.

- Définition de l'ensemble d'entré (l'univers) : Plage de variation du capteur IR droit.
- Variable linguistique : position du robot
- Classes d'appartenance : Trop loin ; Correct ; Trop près
- Fonctions d'appartenance : On décompose la plage d'évolution du robot en plusieurs parties pour appliquer une correction adaptée dans chaque partie :

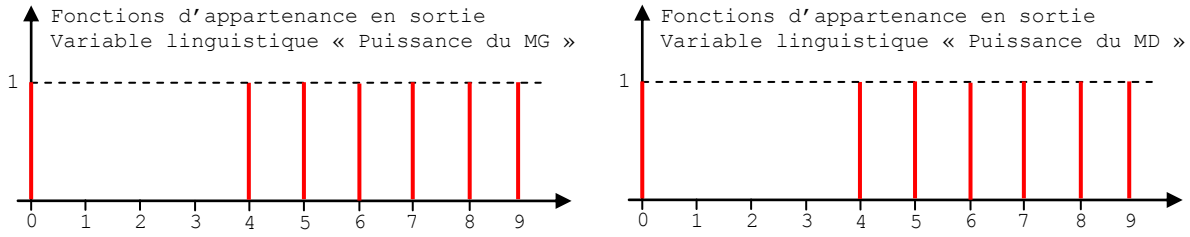


Le système possède 2 variables de sorties : MD (Moteur Droit) et MG (Moteur Gauche).

- Définition des ensembles de sorties (les univers) : Plage de variation de puissance des moteurs MD et MG.
- Variables linguistiques : Puissance du moteur MD ; Puissance du moteur MG
- Classes d'appartenance :

Classes d'appartenance	Puissance associée
STOP	0
VTL (Vitesse Très Lente)	4
VL (Vitesse Lente)	5
VML (Vitesse Moyenne Lente)	6
VMR (Vitesse Moyenne Rapide)	7
VR (Vitesse Rapide)	8
PV (Pleine Vitesse)	9

- Fonctions d'appartenance :



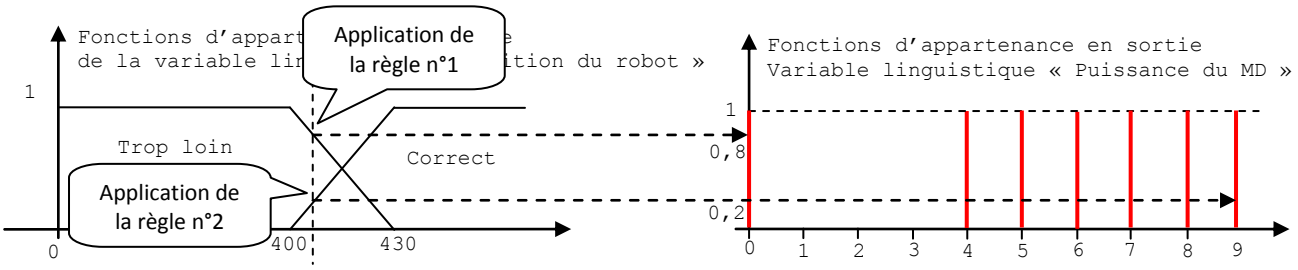
6.6.3 Définitions de règles

N°	Règle
1	**Si** (Position du robot **est** Loin) **Alors** (MG **est** PV) **et** (MD **est** STOP)
2	**Si** (Position du robot **est** Correct) **Alors** (MG **est** PV) **et** (MD **est** PV)
3	**Si** (Position du robot **est** Prêt) **Alors** (MG **est** STOP) **et** (MD **est** PV)

6.6.4 Application des règles et défuzzification

Entre les valeurs 400 et 430, on hésite entre dire que sa position est correct ou trop loin. Les règles n°1 et n°2 sont applicables.

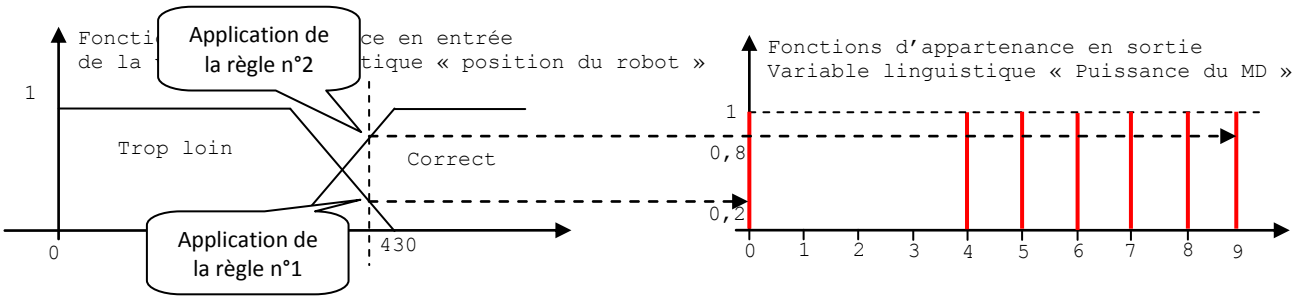
Leur application nous indique que le moteur gauche doit être configuré à pleine vitesse. Pour le moteur droit, on applique les règles ainsi :



Dans ce cas précis, la puissance à appliquer au moteur droit est :

$$MD = 0x0,8 + 9x0,2 = 1,8$$

La puissance a appliqué étant un nombre entier, on appliquera 2.



Dans ce cas précis, la puissance à appliquer au moteur droit est :

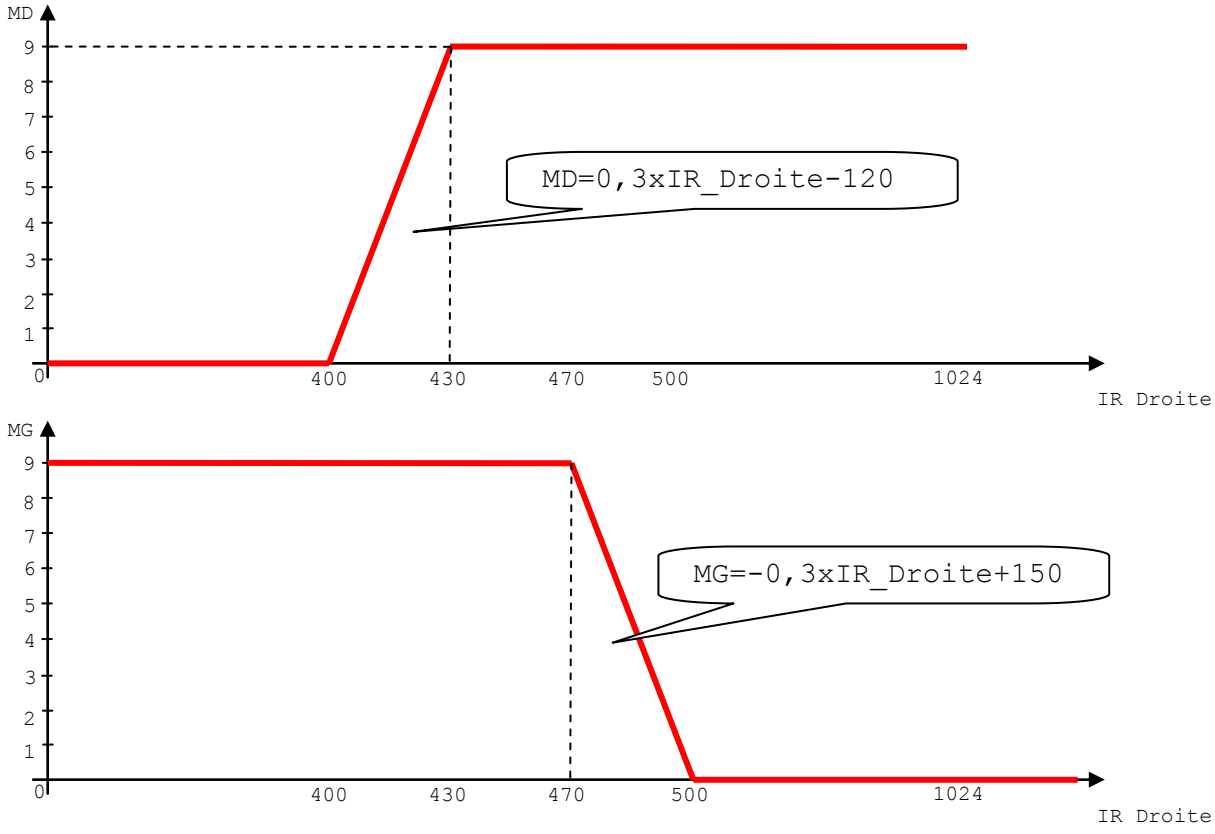
$$\mathbf{MD = 0x0,2 + 9x0,8 = 7,2}$$

La puissance a appliqué étant un nombre entier, on appliquera 8.

Le même résonnement est appliqué sur l'intervalle [470 ; 500] ou les règles n°2 et n°3 s'appliquent.

6.6.5 Conclusion

Les lois de commande des moteurs droit et gauche devraient être :



6.6.6 Programmation sur le KJunior

```
#include "KJunior.h"
void main(void)
{
   signed int8 MD , MG;
   signed int16 IR=0;
   KJunior_init();
   KJunior_config_auto_refresh_sensors(MANUAL);
   while(1)
   {
      KJunior_flag_sensors_reset();
      KJunior_manual_refresh_sensors();
      while(Sensors_Refreshed_Flag == 0); // Wait until the sensor are refreshed

      IR=KJunior_get_proximity(RIGHT);

      if(KJunior_get_proximity(FRONT) > 400) KJunior_set_speed(0,0);
      else if(IR<401)
      {
         MD=0;
         MG=9;
      }
      else if(IR>400 && IR<431)
      {
        MD=(int)(0.3*IR-120);
        MG=9;
      }
      else if(IR>430 && IR<471)
      {
        MD=9;
        MG=9;
      }
      else if(IR>470 && IR<501)
      {
        MD=9;
        MG=(int)(-0.3*IR+150);
      }
      else if(IR>500)
      {
        MD=9;
        MG=0;
      }
      KJunior_set_speed(MG,MD);
   }
}
```

6.6.7 Exemple de correction PID : Azolla

```
function azolla.main(azolla)
        Vmoy=10
        moyenne = 5
        somme=0
        derive=0
        ErreurPrecedente=0
        Kp=9
        Ki=0.01
        Kd=20
        while true do
                front = azolla:readsensor(0)
                right = azolla:readsensor(1)

                delta = right - moyenne
                somme=somme+delta
                derive=delta-ErreurPrecedente
                Vc=Kp*delta+Ki*somme+Kd*derive
                if(front<moyenne) then
                        azolla:setspeed(-5,5)
                else
                        azolla:setspeed(Vmoy+Vc,Vmoy-Vc)
                end
                ErreurPrecedente=delta
                azolla:stepforward()
        end
end
```

6.6.8 Exemple de correction PID : RobotC Lego Mindstorm

```
#pragma config(Sensor, S2,     right,          sensorLightActive)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

float Kp=5;
float Ki=0.005;
int moyenne=30;
int delta;
float somme=0;
int Vmoy=50;
int Vc;

task main()
{
        motor[motorA]=Vmoy;
        motor[motorC]=Vmoy;
        while(true)
        {
                delta = moyenne - SensorValue[right];
                somme = somme + delta;
                Vc = (int)(Kp*delta+Ki*somme);
                motor[motorA]=Vmoy-Vc;
                motor[motorC]=Vmoy+Vc;
        }
}
```

6.7 Circulation dans un couloir

On utilisera deux sonars, un à droite et un à gauche. La meilleure approche de correction de la trajectoire étant le PID, on le programmera sur simulateur et on l'adaptera sur le robot Mindstorm.

```
function azolla.main(azolla)
        KP=2
        KD=5
        KI=0.01
        Vmoy=10
        Vc=0
        Delta=0
        derivDelta=0
        somDelta=0

        while(true) do
                Front = azolla:readsensor(0)
                Fleft = azolla:readsensor(5)
                Fright = azolla:readsensor(1)

                -- Demi-tour si bloqué devant
                if (Front < 5) then
                        azolla:setspeed(4,-4)
                else
                        -- Tolérance d'erreur
                        if(math.abs(Fright-Fleft)<2) then
                                Fright=Fleft
                        end

                        -- Calculs des paramètres de correction
                        oldDelta=Delta
                        Delta = Fright - Fleft
                        derivDelta = Delta - oldDelta
                        somDelta= somDelta + Delta

                        -- Correction de vitesse
                        Vc=(KP*Delta+KD*derivDelta+KI*somDelta)
                        vitMG=Vmoy+Vc
                        vitMD=Vmoy-Vc
                        azolla:setspeed(vitMG,vitMD)
                end
                azolla:stepforward()
        end
end
```

7 Bibliographie

Simulateur Azolla	http://www.codeproject.com/Articles/33587/2D-LUA-Based-Robot-Simulator
Exemples de projets Lego MindStorms	http://www.nxtprograms.com/index2.html
Page listant des ouvrages NXT (ang.)	http://thenxtstep.blogspot.fr/p/books.html
Documents sur le matériel NXT	http://cache.lego.com/upload/contentTemplating/Mindstorms2SupportFilesDownloads/otherfiles/download8CFD37F17F7EFCDC412AE7CEBF245C6A.zip
Labview	http://www.ni.com/labview/f http://www.ni.com/robotics/f http://www.ni.com/academic/mindstorms http://fr.wikipedia.org/wiki/LabVIEW http://labview.developpez.com
RobotC pour Mindstorm	Télécharger – site officiel http://www.robotc.net/ Curriculum – Fiches pédagogiques – activités – Projets – Vidéo de démo http://www.education.rec.ri.cmu.edu/previews/robot_c_products/teaching_rc_lego_v2_preview/
Pour un aperçu très rapide sur la robotique	http://fr.wikipedia.org/wiki/Robot http://fr.wikipedia.org/wiki/Robotique http://fr.wikipedia.org/wiki/Capteur http://fr.wikipedia.org/wiki/Actionneur http://fr.wikipedia.org/wiki/Régulation http://fr.wikipedia.org/wiki/Régulation_automatique http://fr.wikipedia.org/wiki/Logique_floue Émission C'Dans l'air : Les robots sont parmi nous - 15/03/2012
À propos de la logique floue	Introduction à la logique floue - Antoine Cornuéjols (http://www.lri.fr/~antoine/Courses/AGRO/Cours-IA/Tr-logique-flouex4.pdf) Le site du zéro – Introduction à la logique floue http://www.siteduzero.com/tutoriel-3-637002-introduction-a-la-logique-floue.html Introduction à la logique floue - Matthieu Lescieux (Ecole Polytechnique de Tours) http://auto.polytech.univ-tours.fr/automatique/AUA/ressources/ Robot Flou – Article sur le blog Pobot – Robotique Sophia Antipolis http://www.pobot.org/Robot-Flou.html La logique floue - Laboratoire Analyse et Commande des Systèmes (LACS- Tunisie) http://www.tn.refer.org/hebergement/cours/logique_floue/
À propos des correcteurs PID	http://fr.wikipedia.org/wiki/Régulateur_PID http://en.wikipedia.org/wiki/PID_controller New HiTechnic Motor PID Block http://www.hitechnic.com/blog/uncategorized/pid-block/ Forum SeTechnic – Le PID, un contrôleur intelligent http://www.setechnic.com/Forum/topic2770.html Asservissement d'un robot autonome – Club robotique de l'INSA de Lyon http://clubelek.insa-lyon.fr/joomla/fr/base_de_connaissances/informatique/