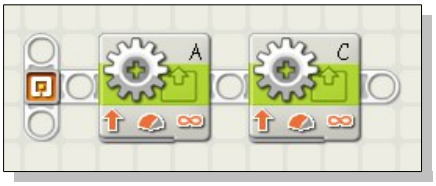
Annexe 1 : mise en œuvre des quatre défis

Avertissement : dans cet exemple, les moteurs seront connectés sur les ports A (moteur droit) et C (moteur gauche) du NXT.
Le capteur optique est connecté sur le port 1, le capteur tactile sur le 2, et le capteur ultra-son sur le 3.

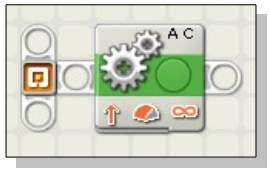
1 / Défi 1 : faire avancer le robot

Les élèves doivent programmer le robot de façon à ce qu'il se déplace en ligne droite, sans limite de distance ni de durée.

Plusieurs programmes sont possibles dans ce cas :



En utilisant un bloc *Moteur* pour chaque moteur.



En utilisant un bloc *Déplacement* unique pour les deux moteurs

Notons qu'il est probable que les élèves ne penseront pas à arrêter les moteurs pour stopper le robot ! Lors de l'exécution de ce programme, les moteurs vont bien démarrer, mais seront stoppés rapidement dès la fin du programme.

L'utilisation d'un bloc *Déplacement* facilite la programmation, en pilotant les deux moteurs en même temps, mais aussi avec la même configuration. Les élèves ne devraient pas avoir de soucis à résoudre la problématique avec ce bloc : les paramètres par défaut permettent de faire avancer le robot, de la distance correspondante à un tour complet de moteur.

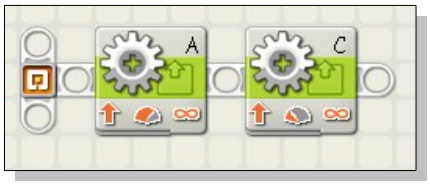
Il paraît plus formateur de choisir l'option avec les deux blocs *Moteur* : chaque organe est géré indépendamment des autres, ce qui permet de régler leurs paramètres propres, et donc de faire réagir le robot d'une façon plus fine.

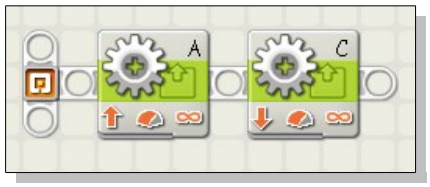
Avec cette solution, les élèves pourront être confrontés à divers problèmes :

- l'un des moteurs n'est pas lancé : le robot tourne sur l'axe de la roue dont le moteur n'est pas alimenté ;

- les deux moteurs n'ont pas la même configuration (puissance, durée d'activation, sens…) : le robot n'ira pas droit, mais tournera vers une direction qui dépendra du problème de configuration.

Il conviendra alors de faire réfléchir les élèves sur les conséquences de chacun de ces cas d'erreur : le résultat de la réflexion pourra être réinvesti par la suite, lorsque nous souhaiterons faire tourner le robot.

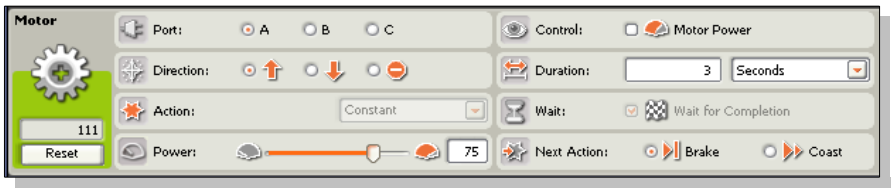




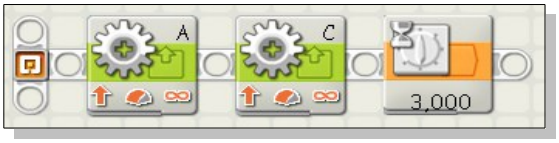
Par la suite, on intègre une contrainte de durée de déplacement : le robot doit avancer en ligne droite pendant trois secondes.

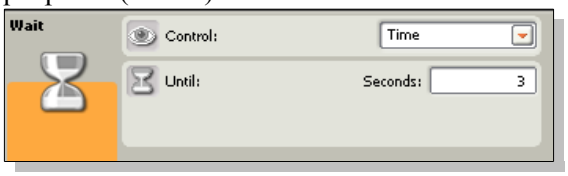
La durée d'activation des moteurs peut être réglée de plusieurs façons :

- directement dans la configuration du bloc *Moteur* ou *Déplacement.* Mais, là, les élèves constateront, avec stupéfaction, que le programme ne fonctionne pas comme ils l'avaient pensé : le robot tourne vers la gauche pendant 3 s, puis vers la droite 3 s aussi. Il apparaît alors que le programme est bien séquentiel : le premier bloc doit se terminer avant de pouvoir exécuter le bloc suivant ;



- en utilisant un bloc *d'Attente.* Cette solution semble répondre parfaitement à la problématique … mais non ! Les élèves constateront que le robot avance plus longtemps que prévu (4 à 5 s).





Les élèves devront alors forcer les moteurs à s'arrêter en utilisant de nouveaux blocs *Moteurs* :

Les 3 secondes seront alors respectées.

Les élèves vont devoir maintenant trouver une solution pour faire avancer le robot sur 50 cm. Plusieurs manières seront proposées :

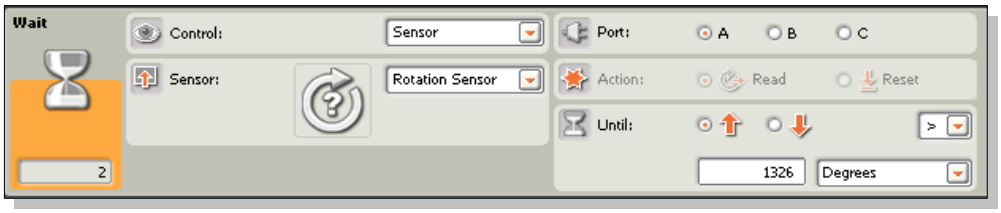
- lancer les moteurs pendant une certaine durée. Mais cela nécessitera de connaître la vitesse du robot en fonction de la puissance, et de la charge appliquées aux moteurs. Les élèves pourront mesurer expérimentalement cette vitesse, et retrouver la durée d'activation des moteurs correspondante ;
- lancer les moteurs en leur faisant faire un certain nombre de rotations (exprimé en nombre de tours complets ou degrés). Connaissant le diamètre des roues, un calcul simple permettra de trouver le nombre exact de rotations.

Par exemple, les roues fournies avec la version NXT 2.0 ont un diamètre de 4,32 cm[1]. Un tour de moteur fait donc parcourir 13,57 cm au robot ($4{,}32\,\text{cm} \times \pi = 13{,}57\,\text{cm}$). Pour avancer de 50 cm, les roues doivent effectuer 3,68 tours, soit 1326,29 degrés. On découvre au passage qu'il n'est pas toujours judicieux d'activer les moteurs à la vitesse maximale, car la phase d'accélération n'est plus du tout négligeable si on va trop vite.[2]

Le programme précédent peut être aisément ici adapté : seul le bloc *d'Attente* est reconfiguré :

1 Quand elles sont neuves !

2 Ou alors, si on veut vraiment aller vite, il faut prévoir une « montée en vitesse » des deux moteurs et une descente analogue, ce qui entraîne des calculs un peu complexes (ou la mise au point de tables).



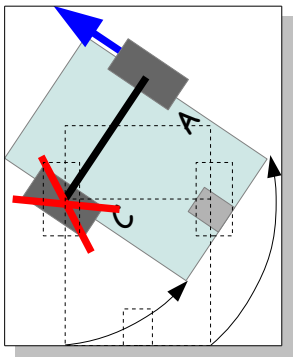
2 / Défi 2 : faire tourner le robot

Les élèves doivent programmer le robot de façon à ce qu'il vire d'un côté ou de l'autre, sans limite angulaire.

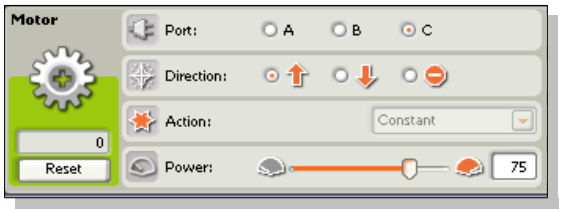
Le but de cette première démarche est purement qualitatif ou phénoménologique : qu'observe-t-on ?

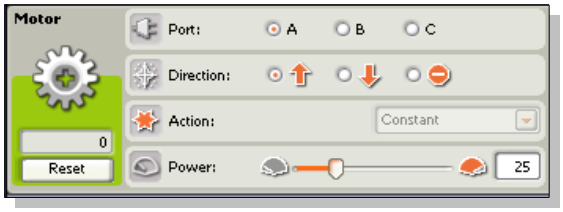
Le programme finalisé précédemment reste valide car seules les configurations des deux premiers blocs *Moteurs* devront être ajustées en fonction des cas.

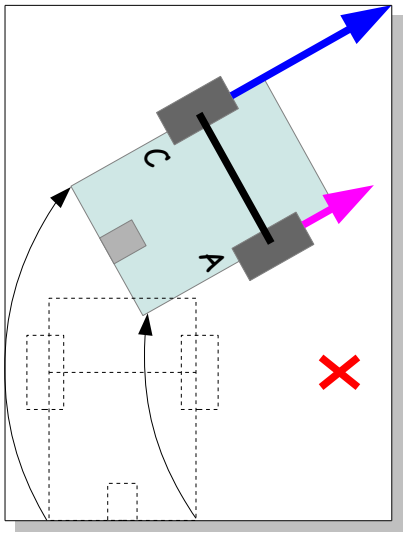
– l'un des moteurs (A) est activé mais pas l'autre (C) : le robot tourne autour de la roue immobile (la roue gauche) ;



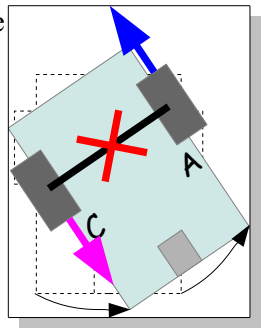
– les deux moteurs ont des configurations de puissance différentes (25 % pour le moteur A, 75 % pour le moteur C) : le robot tourne en arc de cercle et vire du côté du moteur ayant la plus faible puissance (vers la droite). La position du centre de rotation dépend des puissances renseignées ;







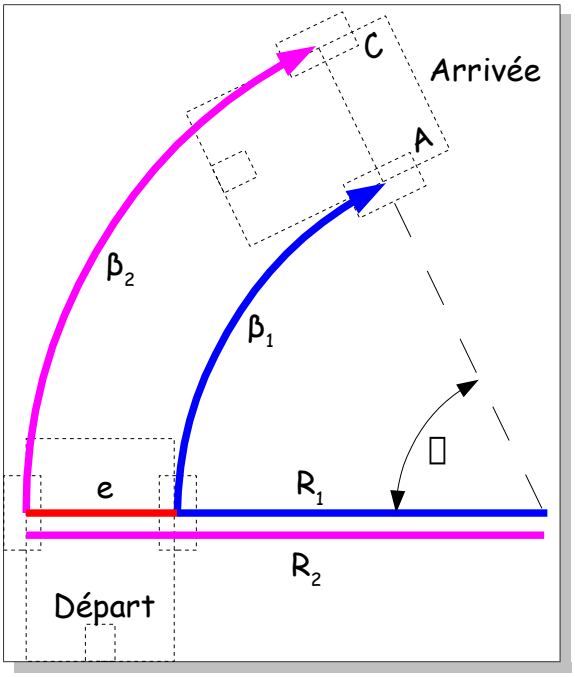
- les deux moteurs sont activés à la même puissance mais ne tournent pas dans le même sens : le robot tourne autour du milieu de l'axe de transmission des roues et vire dans la direction de la roue qui tourne vers l'arrière du robot.



3 / Défi 3 : Rotation du robot autour d'un point

3.1 Position du problème

On souhaite faire faire au robot une rotation vers la droite d'un angle α, typiquement un quart de tour. Comment procéder ? Et où le robot sera-t-il à l'issue de ce déplacement ?



3.2 Actionneurs et capteurs

Chaque moteur peut être actionné à une certaine puissance, définie en pourcentage de la puissance maximale, correspondant à une vitesse de rotation (non précisée). Par ailleurs, un capteur (circuit intégrateur nommé « Rotation sensor ») surveille en permanence, pour chaque moteur, l'angle de rotation à partir de la position initiale du mouvement.

Il n'est pas possible de se baser sur la vitesse des roues parce que la mise en marche ne peut physiquement pas être instantanée (même si on parvenait à passer subitement de la vitesse zéro à la vitesse maximale, cela entraînerait un glissement au niveau des roues donc une perte de précision). Pareillement, le freinage ne peut pas être instantané.

3.3 Géométrie et notations

Nous admettons (cela est assez évident et a été constaté lors des défis précédents) que le robot décrit un arc de cercle lorsque les deux moteurs sont activés à des puissances différentes.

La roue gauche, correspondant au moteur activé à la plus grande puissance, décrit un cercle de rayon R_2 et la roue droite décrit un cercle de rayon R_1 ; l'écartement des roues, constant, est $e = R_2 - R_1$ [3]. Les roues sont toutes deux de rayon r, la roue gauche tourne sur elle-même d'un angle β_2 et la roue droite d'un angle β_1, ce sont des quantités « observables » grâce aux capteurs angulaires. Enfin, si le moteur de la roue gauche a une puissance p_2 et celui de la roue droite une puissance p_1, notons $\mu = \frac{p_1}{p_2}$ le rapport des puissances.

En somme, dans ce problème, nous avons des données (ou contraintes) e, r, α, μ, et des inconnues $\beta_1, \beta_2, R_1, R_2$.

β_2	la distance que la roue C, ayant le plus de puissance, doit parcourir en cm ;
R_2	la distance entre la roue C et l'axe de la rotation du robot en cm ;
β_1	la distance que la roue A, ayant le moins de puissance, doit parcourir en cm ;
R_1	la distance entre la roue A et l'axe de la rotation du robot en cm ;
e	la distance séparant les deux roues du robot. Dans notre cas, e = 12,7 cm ;
α	l'angle de rotation du robot, exprimé en degrés.
r	le rayon de la roue. Dans notre cas, r = 2,16 cm [4]

3 Cette donnée dépend du montage de robot choisi. Elle est fortement variable.

4 Cette valeur est liée au type de la roue utilisée et peut être adaptée en fonction.

3.4 Un peu de physique

Nous allons supposer que la liaison entre les roues et le sol se fait sans glissement ; cela impose que si le sol soit légèrement rugueux ou en tous cas pas trop lisse, et que les accélérations communiquées aux roues ne soient pas trop fortes (comme pour les « vraies » voitures qui se mettent à déraper lorsqu'on appuie trop vigoureusement sur la pédale d'accélérateur). De la sorte, nous avons une relation de proportionnalité entre la distance parcourue par le point de contact de la roue sur le sol, et l'angle (total) de rotation de la roue, soit : $r\beta_2 = R_2\alpha$ et analogue pour l'autre roue.

Nous supposerons aussi que la puissance appliquée aux moteurs est toujours proportionnelle à la vitesse angulaire de ceux-ci, ce qui suppose une certaine limitation qui n'est absolument pas gênante (aller trop vite signifie perdre de la précision). On aura donc une relation de proportionnalité entre les vitesses angulaires : $\frac{d\beta_1}{dt} = \mu \frac{d\beta_2}{dt}$ ou encore $\frac{d}{dt}(\beta_1 - \mu\beta_2) = 0$. Comme les valeurs initiales des angles β_1 et β_2 peuvent être prises nulles, on a la relation (assez intuitive) : $\beta_1 = \mu\beta_2$.

3.5 Un peu de calcul

La relation géométrique $e = R_2 - R_1$ amène $e\alpha = (R_2 - R_1)\alpha = r(\beta_2 - \beta_1)$ soit $\beta_2 - \beta_1 = \frac{e\alpha}{r}$.

3.6 L'algorithme de pilotage du robot

Si nous voulons simplement provoquer une rotation d'angle α, la formule précédente nous suggère de piloter le robot en surveillant la différence $\beta_2 - \beta_1$ sans avoir à tenir compte les puissances appliquées.

- activer le moteur de gauche à la puissance p_2 et le moteur de droite à la puissance $p_1 = \mu\, p_2$.[5]
- surveiller $\beta_2 - \beta_1$
- arrêter les deux moteurs dès que $\beta_2 - \beta_1 = \frac{e\alpha}{r}$ (on doit évidemment mesurer tous les angles dans la même unité, le degré convenant tout à fait ici).

3.7 Et le point d'arrivée ?

On a une proportionnalité entre le rayon et la longueur d'arc sur le cercle, si bien que nous avons aussi $R_1 = \mu R_2$ à partir de $\beta_1 = \mu\beta_2$. La relation $R_1 = R_2 - e$ amène donc $\mu R_2 = R_2 - e$ soit $R_2 = \frac{e}{1-\mu}$. En combinaison avec l'angle α, la connaissance de R_1 et R_2 permet de savoir où se trouve le robot à l'issue du mouvement.

3.8 Un cas particulier simple mais peu recommandé

En faisant tendre le coefficient μ vers 0, la puissance p_1 et le rayon R_1 tendent vers 0. Tout se passe comme si la roue droite était immobile, la roue gauche décrivant un arc de cercle de rayon $R_2 = e$ et d'angle $\beta = \frac{e\alpha}{r}$. La situation semble plus simple, mais elle ne convient guère car la roue droite n'est pas si immobile que cela : elle pivote sur elle-même autour d'un axe vertical, et effectue donc un léger glissement ; il n'est pas du tout certain que le centre instantané de rotation dans ce mouvement soit fixe[6], si bien que la précision ne sera pas bonne.

3.9 Un cas particulier simple et très recommandé

Nous pouvons enfin envisager de faire tourner les deux roues en sens inverse mais avec des puissances égales. Il est alors à peu près évident que le robot pivote (d'un angle α) autour du milieu entre les points de contacts des deux roues (voir haut de page 4). Si on imagine un pivot virtuel situé à cet endroit, on comprend que c'est exactement le même cas que précédemment en remplaçant e par $\frac{e}{2}$! On a donc en ce cas $\beta_1 = \beta_2 = \frac{e\alpha}{2r}$, ce qui

5 Il faut ici éviter de choisir μ trop proche de 0 ou de 1 : si μ est très petit on est dans le cas particulier qui va être étudié plus loin et la roue glisse ; et si μ est proche de 1, la trajectoire du robot est quasiment rectiligne, et celui-ci va sortir de la zone d'évolution avant que la rotation espérée n'ait pu être achevée.

6 Il y a des réactions à prendre en compte, car le centre d'inertie du robot n'est pas du tout à la verticale de la zone de contact de la roue immobile.

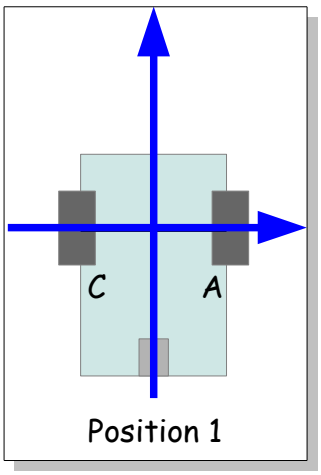
donne lieu à un algorithme très simple :

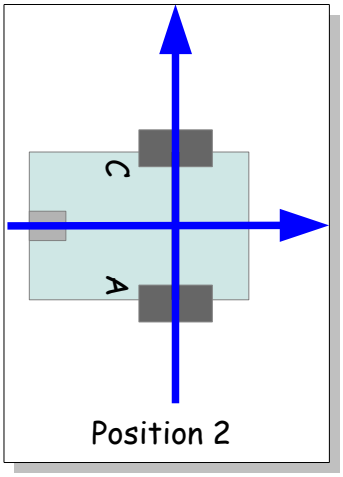
- activer le moteur de gauche à la puissance p_2 et le moteur de droite à la puissance[7] $p_1 = -p_2$.
- surveiller β_1 ou β_2
- arrêter les deux moteurs dès que $\beta_2 = \frac{e\,\alpha}{2r}$.

Un avantage de cette manière de procéder est que le robot ne se déplace plus beaucoup, ne risquant guère de rencontrer d'obstacle ; ainsi sa position à l'issue du mouvement est aisément prévisible !

3.10 Contrainte angulaire de 90°

À ce moment-là, on peut demander aux élèves de faire tourner le robot de 90° dans le sens horaire. Pour commencer, on peut les interroger sur les possibilités d'arriver de la position 1 à la position 2 comme indiqué ci-dessous.





Soit le robot a viré de 90° dans le sens horaire, soit il a parcouru 270° dans le sens anti-horaire !

Plusieurs solutions sont valides : les élèves doivent être conscients qu'il faut choisir celle qui conviendra au problème en cours.

Nous avons vu, par les exercices précédents, qu'il existe diverses méthodes pour faire virer le robot. En utilisant chacune d'entre elles, les élèves trouvent les paramètres à appliquer aux blocs *Moteurs* et/ou *Attente* pour faire virer le robot de 90°. Chacune des deux stratégies ci-dessous a ses avantages et domaines d'application (suivant qu'on se déplace dans un espace de type « plan quadrillé avec obstacles » ou « couloirs »).

Les deux moteurs ont des configurations de puissance différentes.

Supposons les valeurs de puissance : 25 % pour le moteur A, 75 % pour le moteur C. Le robot tourne en arc de cercle et vire du côté du moteur ayant la plus faible puissance (vers la droite). La position du centre de rotation dépend des puissances renseignées.

L'algorithme décrit au chapitre 3.6 - L'algorithme de pilotage du robot (page 5) se termine lorsque $\beta_2 - \beta_1 = \frac{e\,\alpha}{r} = \frac{12{,}7 \times 90}{2{,}16} = 529{,}16$ (en degrés).

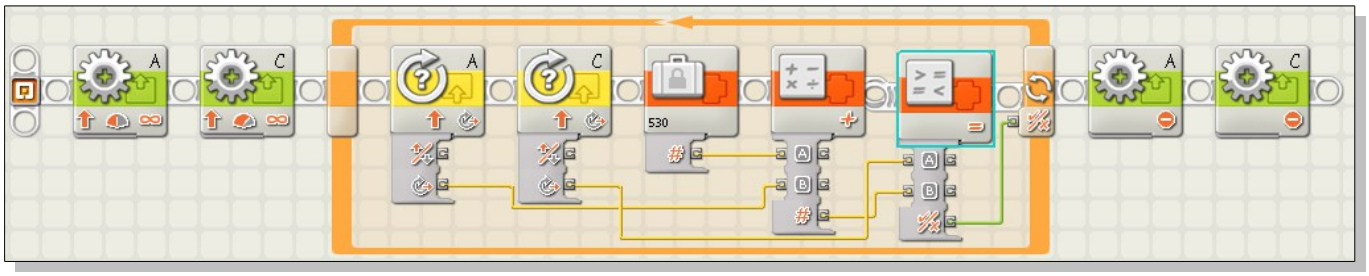
> Il est important de remarquer ici que les blocs de programmation Lego MindStorms NXT n'acceptent pas les valeurs numériques réelles. En fonction des résultats, il est nécessaire d'effectuer des arrondis plus ou moins arbitraires, mais engendrant des erreurs de positionnement. Le « rattrapage » des erreurs d'arrondi sera nécessaire si de nombreuses rotations sont à prévoir, ce qui va se produire dans le problème du parcours de labyrinthe (Annexe 2).

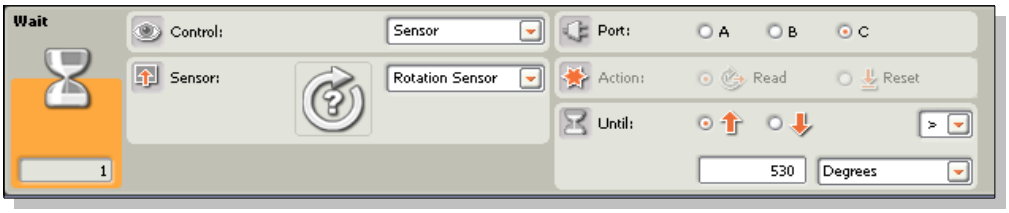
Les capteurs de rotation de chaque moteur vont être scrutés en permanence. Les valeurs retournées, représentant le nombre de degrés que chaque moteur a effectué depuis son lancement, sont alors comparées. Dès que la différence du nombre de degrés effectués par le moteur C (β_2) et celui du moteur A (β_1) égale 530, les deux moteurs seront arrêtés. Si tout va bien, le robot a viré de 90° !

Le programme à réaliser fait intervenir une boucle JUSQU'A pour scruter les capteurs de rotation, ainsi que des

7 Le signe négatif indique un sens opposé pour les moteurs, tout en gardant une valeur de puissance identique.

blocs *Constante*, *Mathématique* et *Comparaison* simples.



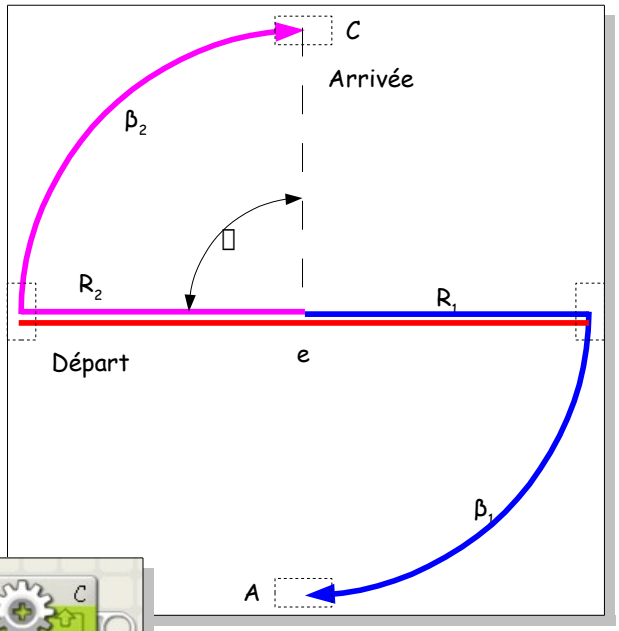


Un programme bien plus paramétrable pourrait faire l'objet d'un approfondissement : intégrer la formule générale, et proposer trois variables à renseigner : α, e et r.

Les deux moteurs ne tournent pas dans le même sens.

Le robot tourne autour du milieu de l'axe de transmission des roues, et vire dans la direction de la roue qui tourne vers l'arrière du robot. Ici le problème se résout aisément.

Soit le schéma ci-contre, représentant les positions de départ et d'arrivée du robot, ainsi que les cotations à prendre en considération.

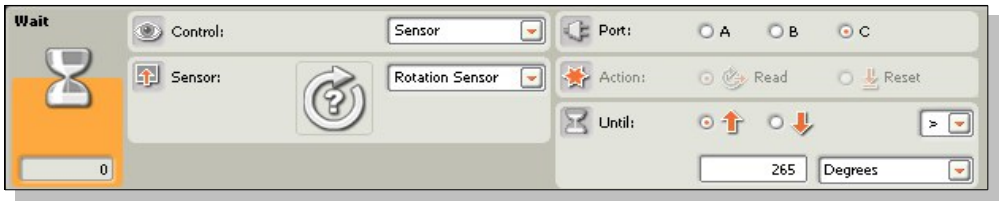


D'après ce qui a été exposé au chapitre 3.9 - Un cas particulier simple et très recommandé (page 5), on va scruter les moteurs afin de détecter à quel moment on a

$$\beta_1=\beta_2=\frac{e\,\alpha}{2r}=\frac{12{,}7\times 90}{2\times 2{,}16}=264{,}58 \text{ (en degrés).}$$

On doit donc attendre que les moteurs aient tourné de 265° pour les arrêter !

Le programme est alors, avec sa configuration :



4 / Défi 4 : détourner le robot des obstacles

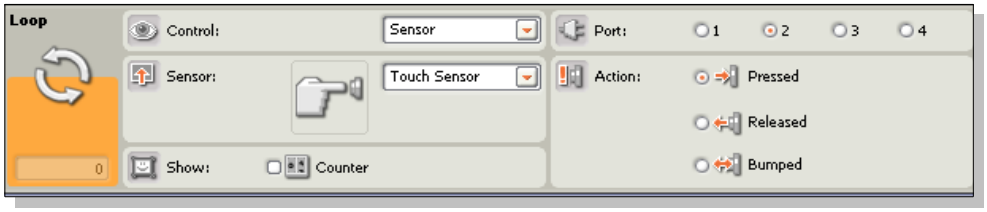
Dans un premier temps, on demande aux élèves de choisir un des capteurs disponibles, permettant de renseigner le robot sur son environnement direct ; dans le cadre de notre problème, bien sûr.

Il apparaît que les capteurs tactile et ultra-sonique sont les plus adaptés.

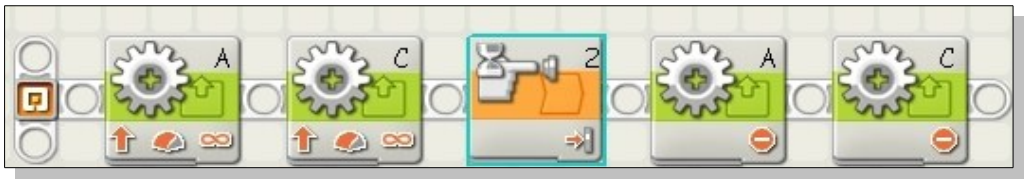
4.1 Capteur tactile

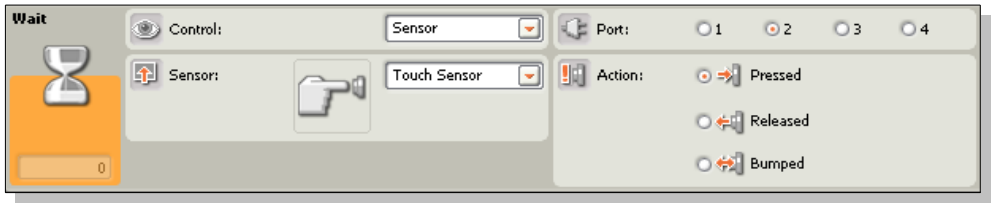
À partir du capteur tactile, par exemple, l'enseignant demande de reprendre le programme réalisé pour faire avancer le robot tout droit, et de le modifier pour que le robot s'arrête en fonction de l'état de son capteur.

Le programme généré ressemble alors au suivant, utilisant une boucle :

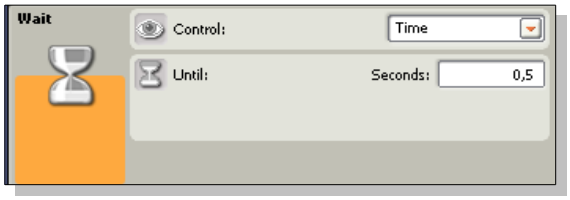


ou bien en utilisant un bloc *Attente* :





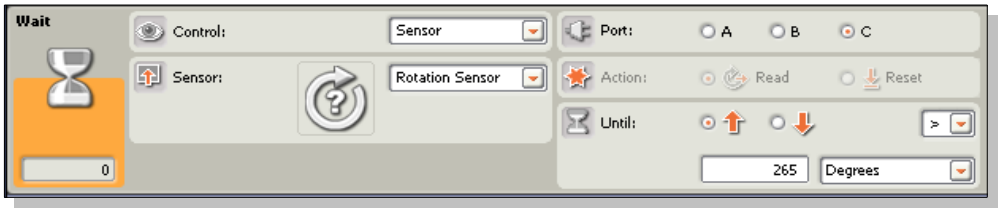
Dans quelques cas particuliers, il se trouve que le robot ne peut tourner sans risquer d'endommager son capteur ou une autre de ses pièces proéminentes. On veillera à le faire reculer un peu pour qu'il se dégage et pallier d'éventuels problèmes :



Ici encore, plusieurs solutions sont possibles. La mise en œuvre d'un bloc *Attente* temporel est présentée à cette occasion.

Par la suite, le robot tourne d'un quart de tour. Les élèves peuvent choisir la méthode de leur choix puisqu'ils ont dû les étudier précédemment. Il est toutefois préférable de faire pivoter le robot sur lui-même : cette technique permet de limiter l'espace utilisé par le robot pour faire sa manœuvre.

La suite du programme est alors :



Le robot a évité ce premier obstacle… Il peut maintenant repartir vers de nouvelles embûches droit devant lui !

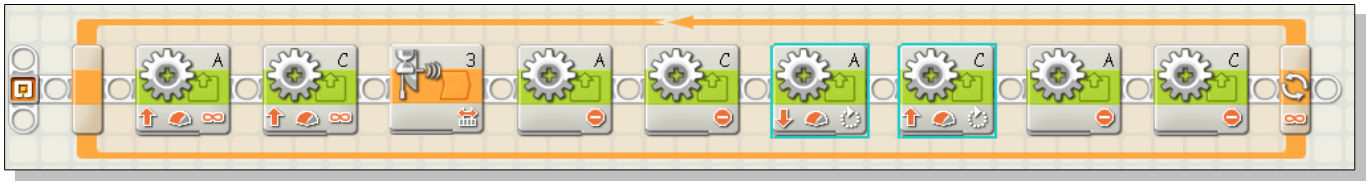
Le programme complet est placé dans une boucle infinie.

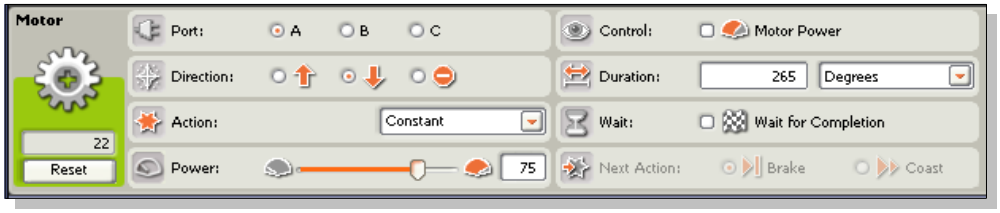
Le terrain de test devra mettre en évidence les divers problèmes pouvant survenir : au moyen de plusieurs types d'obstacles, le robot réagit correctement ou non. On pourra utiliser un mur (parfait), une bouteille de verre (bon), une canette en aluminium (moyen : trop léger), un écran de papier ou de tissu (dépend de la tension et de la solidité), un stylo tenu verticalement (bon s'il est touché par le capteur) ou encore une boîte d'allumettes (mauvais car trop légère, même pleine).

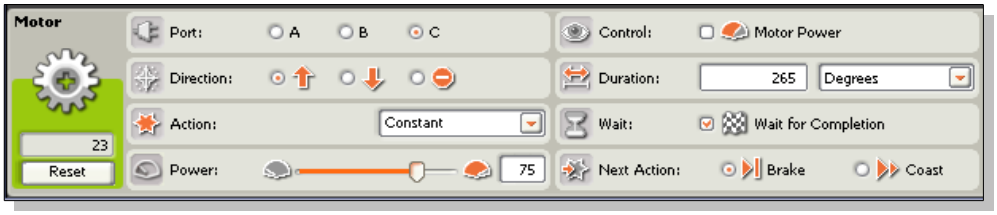
4.2 Capteur ultrasonique

Par la suite, le même programme est adapté pour répondre à l'utilisation du capteur ultra-son. Dans le principe, rien ne change, si ce n'est que le robot n'a plus besoin de reculer avant de tourner : le capteur ultra-son peut détecter l'obstacle à une distance raisonnable et sans contact !

Une des propositions serait :

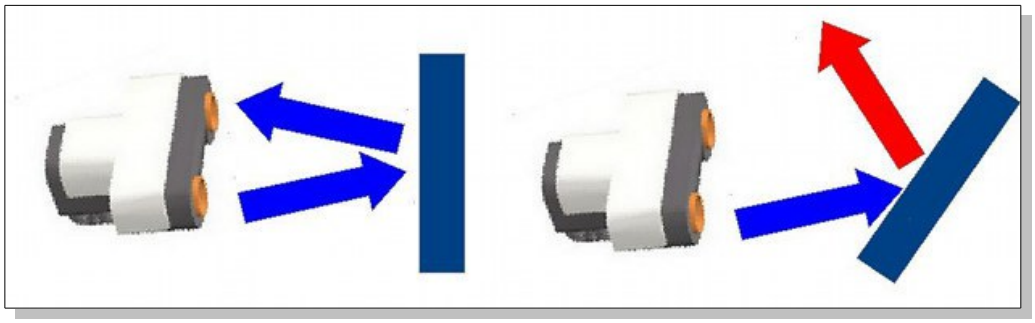




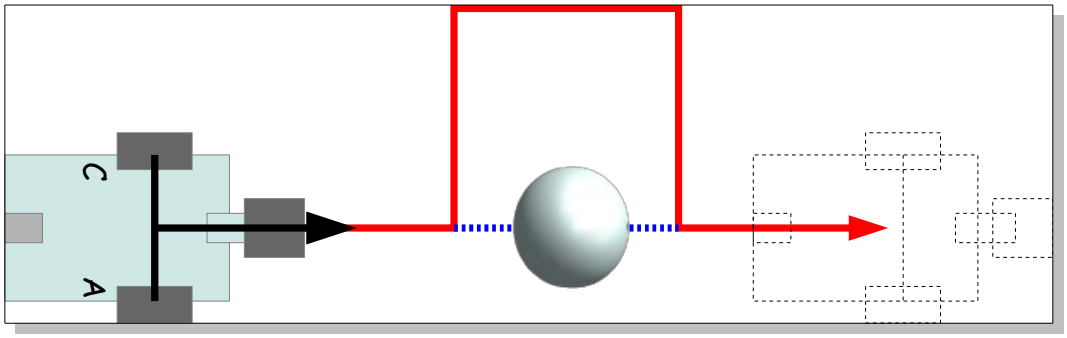


Ici aussi, le terrain de test utilise plusieurs types d'obstacles que le robot devra éviter. On pourra utiliser un les mêmes que précédemment avec une petite différence de réaction pour le stylo tenu verticalement (trop fin pour être détecté) et la boîte d'allumettes (le résultat dépend de son orientation).

Dans tous les cas, il convient de mettre en évidence aussi que l'obstacle n'est détecté que s'il se présente directement face au capteur ; c'est-à-dire à une hauteur raisonnable pour qu'il soit perçu. En effet, l'angle de réflexion du signal ultrason émis par le capteur doit permettre la réception de son écho.



Finalement, les élèves contournent entièrement l'obstacle, en mettant en œuvre une succession de pivotages à 90° et d'avances rectilignes, de façon à ce que le robot récupère sa trajectoire rectiligne initiale.



5 / Défi 5 : faire suivre une ligne au robot

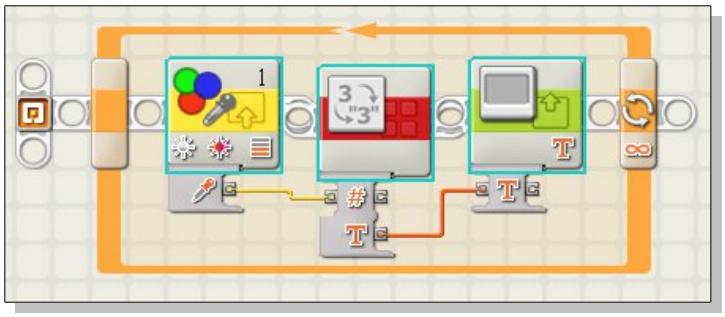
Remarque pour ce défi, et quelle que soit la méthode utilisée.
Il est nécessaire, à chaque séance, voire à divers moments de la journée, de reprendre la mesure des teintes claires et sombres : la luminosité ambiante peut varier de façon significative et donc fausser l'exécution du programme du robot.

5.1 Méthode par seuillage discriminant

Pour établir le seuil, il convient de relever la valeur retournée par le capteur de couleur alors que le robot est placé sur un fond clair, puis sur un fond sombre.

Le capteur nous retourne potentiellement une valeur comprise entre 0 et 100, 0 représentant le noir le plus complet tandis que 100 correspond au blanc le plus pur.

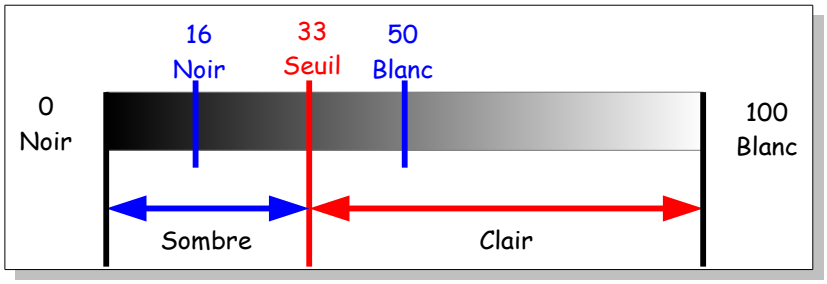
Les élèves commencent donc par écrire un programme qui affiche en boucle, sur l'écran du NXT, la valeur reçue de ce capteur.



Dans un contexte de mise au point, on obtient la valeur 16 pour le noir et la valeur 50 pour le blanc. Donc, toutes les valeurs inférieures ou égales à 16 correspondent à une teinte sombre, toutes celles qui sont supérieures ou égales à 50 représentent une teinte claire. Mais qu'en est-il des valeurs intermédiaires, comprises entre 17 et 49 inclus ?

On va donc couper la plage problématique, en effectuant une moyenne algébrique, de façon à définir un seuil au dessus duquel les valeurs indiqueront les teintes claires, et en dessous duquel seront les teintes sombres.

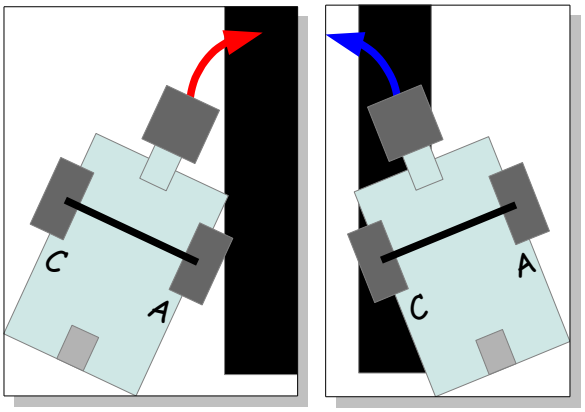
Le seuil est alors calculé à la valeur 33.



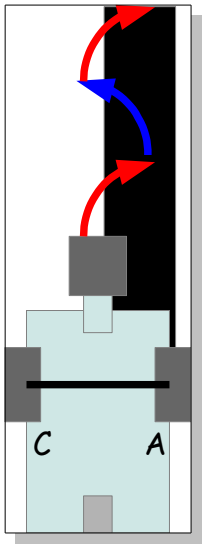
En gardant en tête cette notion de seuil, nous allons demander aux élèves d'écrire un algorithme permettant de faire suivre la ligne noire au robot.

Certainement que les propositions seront diverses, mais on peut guider leur réflexion, en rappelant les différents exercices et résultats passés : faire avancer et virer le robot. Les élèves devront réinvestir ces connaissances.

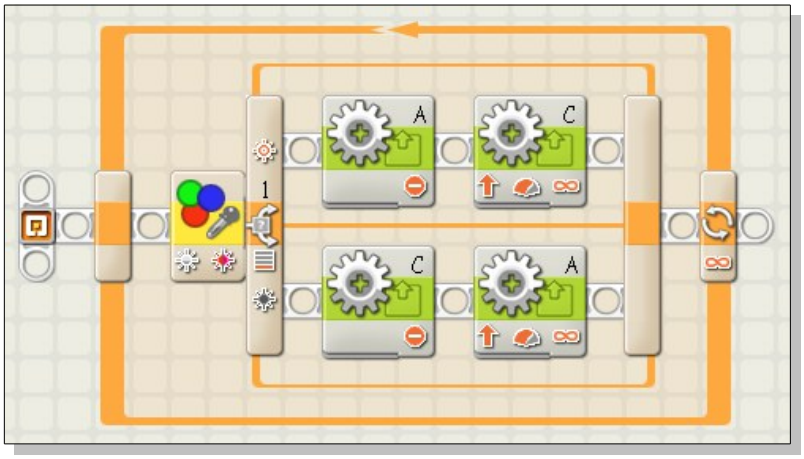
Le principe de base retenu ici, est de faire tourner le robot dans une direction donnée (disons, la droite) si le capteur détecte une teinte claire ; et dans la direction opposée (donc la gauche ici), pour la détection d'une teinte sombre.



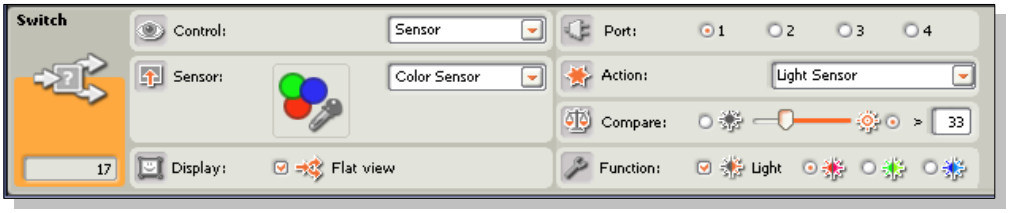
Le robot est placé de façon à ce que le capteur ne soit pas trop éloigné de la ligne à suivre, voire sur la ligne elle-même. Dès le lancement du programme, le robot va tourner dans le sens correspondant à la teinte sur lequel il est.



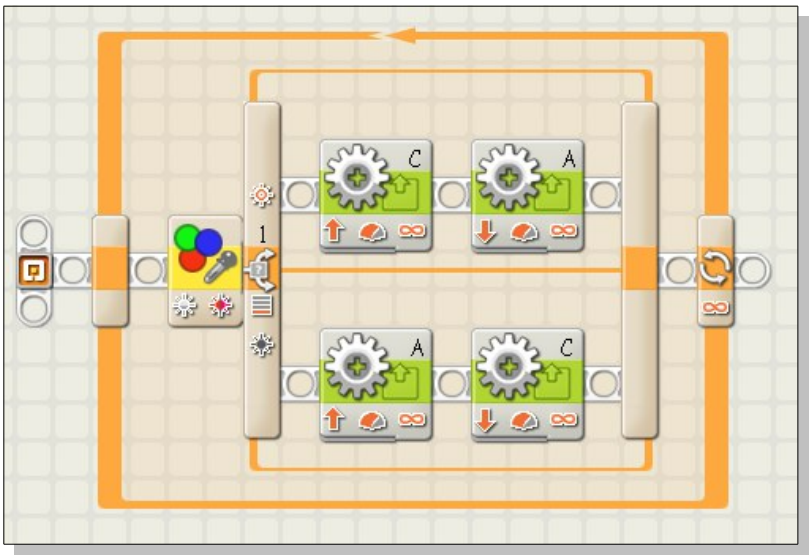
Il se trouvera un moment où le capteur détectera la teinte opposée : il virera dans l'autre sens. Ainsi, par virages successifs et relativement faibles, le capteur du robot sort et rentre dans la bande noire ; le robot « godille » alors sur la bordure gauche de la bande noire !



Un des premiers programmes possibles, qui met en œuvre un bloc *Alternative*, serait alors celui qui figure ci-contre.



Les élèves pourront s'apercevoir que le robot godille assez fortement avec cette solution. Ils pourront alors essayer d'utiliser une des autres façons de faire virer le robot, tout en gardant la configuration du bloc *Alternative* :

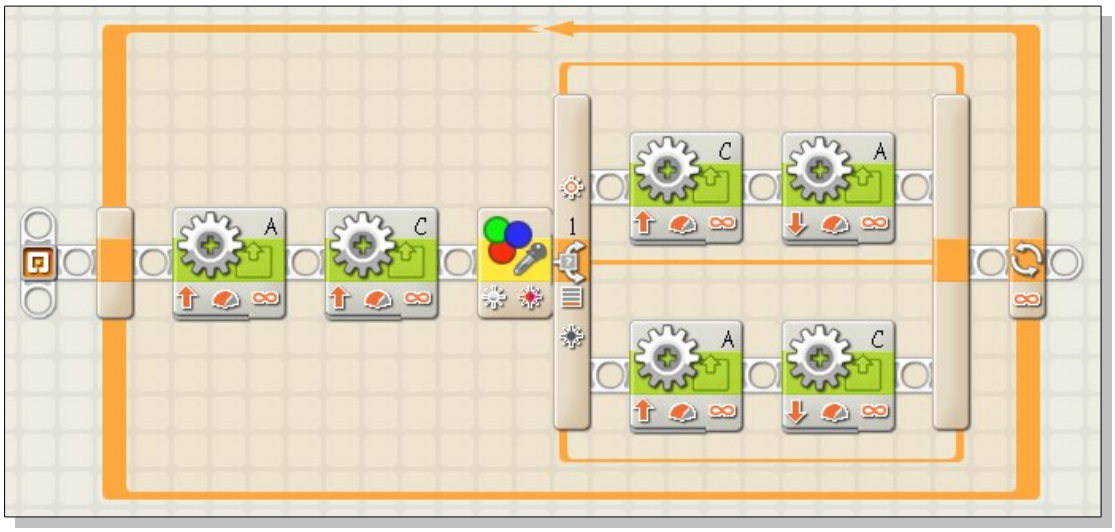


Mais cette solution n'est absolument pas fonctionnelle : le robot n'avance pas !

En effet, il se contente de pivoter sur lui-même (autour du milieu de l'axe de transmission des roues) !

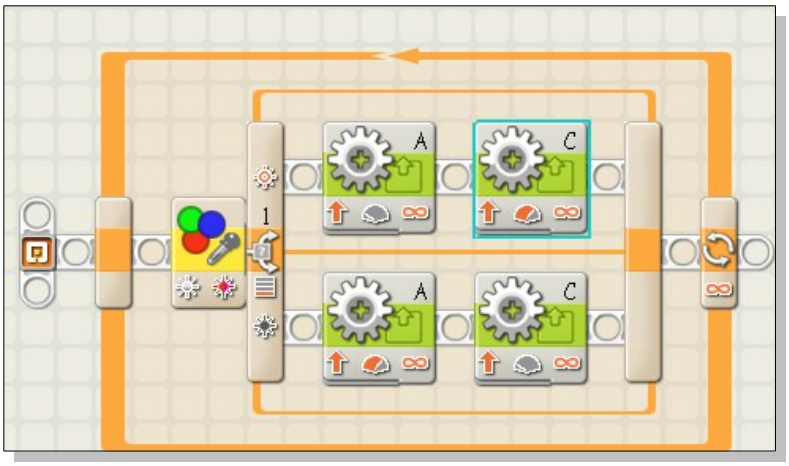
On doit donc ajouter des blocs *Moteurs* permettant de le faire avancer.

Le programme devient alors celui présenté ci-dessous :



Cette proposition ne résout pas le problème de godille du robot, et elle est même assez dangereuse à utiliser : elle peut provoquer des dommages matériels à cause des changements, relativement rapides, de sens de rotation des moteurs !

La dernière solution, qui consiste à appliquer des puissances différentes aux moteurs, paraît alors plus sécurisée pour le matériel, mais qu'en est-il de la réaction du robot ? Godille-t-il moins ?



Ici, on applique des puissances de 10 % et 80 %.

On constate que les courbes de la bande noire sont correctement suivies, mais toujours avec une godille non négligeable.

Les lignes droites génèrent elles aussi de la godille !

Si nous testons des puissances avec un écart moindre, par exemple 45 % et 55 %, on constate que le robot ne godille quasiment plus sur les lignes droites, mais a une inertie assez importante pour les suivre (en fait la fréquence de la godille est seulement plus faible) ! Par contre, le robot n'est plus capable du tout de calquer les courbures !

Quelle est alors la meilleure solution à employer pour limiter la godille du robot ?

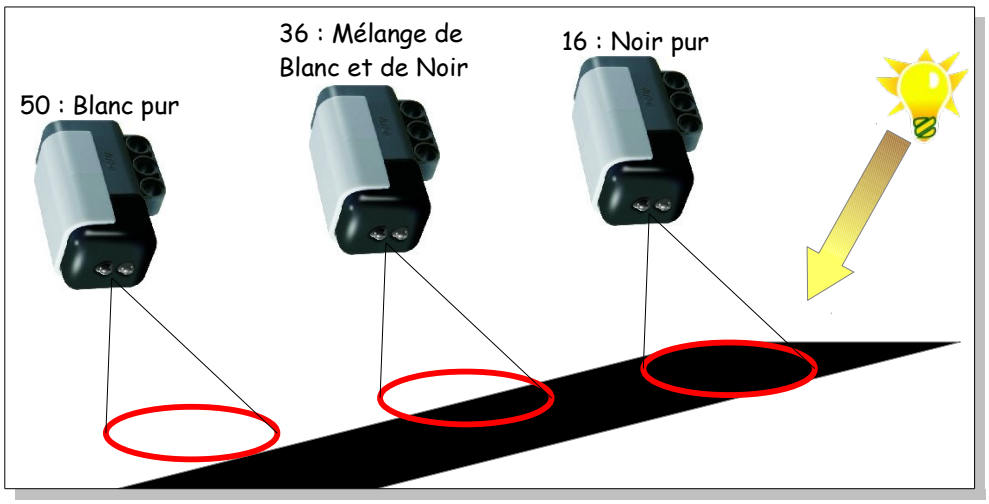
- Mettre deux capteurs optiques côte à côte[8] ? Réduire ou allonger la distance entre le capteur optique et l'axe des roues ?
- Utiliser un algorithme plus pointu : introduction à la logique floue (voir le chapitre 5.2-Méthode par logique floue page 13)?
- Introduire une régulation plus fine ? C'est-à-dire, un PID (voir les références à la fin du document principal) ?

5.2 Méthode par logique floue

On vient de voir que la technique de seuillage n'est pas des plus performantes. Il est possible d'améliorer la précision du suivi de la ligne en introduisant de la logique floue.

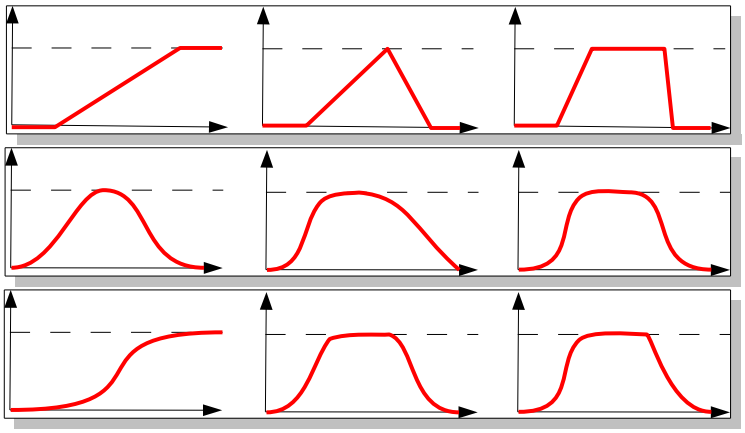
Il s'agit de ne plus faire une simple dichotomie binaire entre le noir et le blanc (seuillage), mais de penser que des nuances de gris sont possibles entre ces deux tons. En effet, par réflexion de la lumière, le capteur indiquera une valeur intermédiaire entre le 16 et le 50, selon son emplacement sur la bordure de la ligne.

Dans un contexte de logique floue, la modélisation de l'évolution des valeurs retournées par le capteur optique, lors de la transition entre le blanc et le noir de notre ligne, est fournie par la ***fonction d'appartenance***. Cette fonction peut revêtir plusieurs forme, selon le problème à traiter, et dans un cadre plus général (source : Frédéric Sur) :

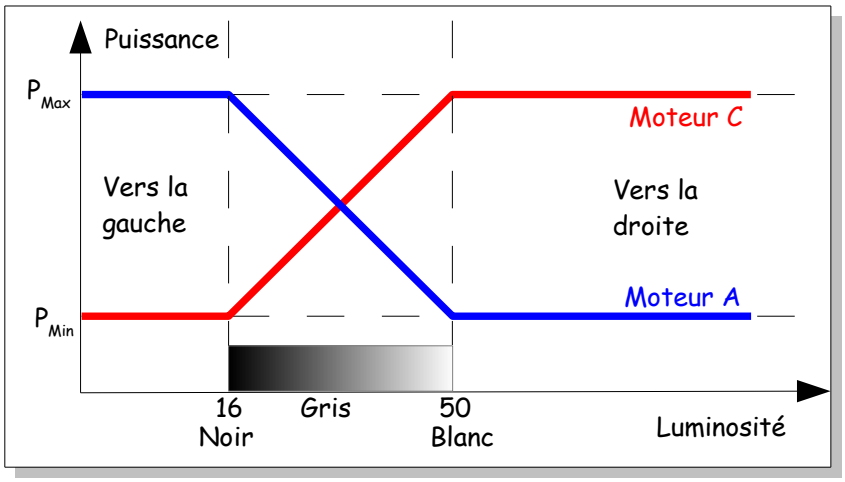


8 C'est la philosophie des véhicules de Braitenberg, voir les références.

- linéaire, triangulaire, trapézoïdale :
- gaussienne :
- sigmoïde :



Dans notre cas, une fonction d'appartenance linéaire donne déjà de bons résultats. On module la puissance à appliquer à chacun des moteurs, en appliquant la fonction d'appartenance suivante :



Le principe de déplacement du robot reste identique : si une couleur sombre est détectée, le robot tourne vers la gauche, sinon il tourne vers la droite.

Mais le robot tourne plus ou moins fortement dans la direction adéquate, selon la valeur reçue du capteur optique. Par exemple, plus la valeur est proche de 16 (le plus sombre possible), plus le robot tournera vers la gauche (le moteur C à la puissance maximale et le moteur A à la puissance minimale).

À l'extérieur de la plage [16;50], les puissances P_{Min} et P_{Max} sont théoriquement utilisées. En effet, le capteur ne peut retourner de valeur plus petite que 16, puisqu'elle correspond au plus sombre possible. De même, et selon le même raisonnement, une valeur supérieure à 50 n'est pas possible !

Dans la plage [16;50], les puissances des moteurs répondent aux fonctions d'appartenance linéaires, que nous allons calculer maintenant.

Pour le moteur A, la fonction d'appartenance liant la puissance $\boldsymbol{P_a}$ et le niveau de gris $\boldsymbol{n}$ est de la forme :

$P_a(n)=a_a\times n+b_a$, a_a et b_a étant deux constantes à calculer, et de même pour le moteur C : $P_c(n)=a_c\times n+b_c$. Il sera également commode de remarquer que dans notre modèle la somme des deux puissances est constante.

En introduisant les puissances maximale et minimale, $\boldsymbol{P_{Max}}$ et $\boldsymbol{P_{Min}}$, à utiliser, et les valeurs de luminosités mesurées pour le noir et pour le blanc, respectivement $\boldsymbol{N}$ et $\boldsymbol{B}$, on a un système d'équations portant sur $\boldsymbol{a_a}$ et $\boldsymbol{b_a}$:

$\begin{cases} P_{Min}=a_a\times N+b_a \\ P_{Max}=a_a\times B+b_a \end{cases}$ ce qui amène par soustraction membre à membre $a_a=\dfrac{P_{Min}-P_{Max}}{N-B}$ (qui est négatif) ; en combinant les deux équations avec des facteurs respectifs $-B$ et N on obtient $b_a(N-B)=P_{max}\times N-P_{Min}\times B$ soit

$b_a = \frac{P_{Max} \times N - P_{Min} \times B}{N - B}$. Finalement, on trouve $P_a(n) = \frac{P_{Min} - P_{Max}}{N - B} \times n + \frac{P_{Max} \times N - P_{Min} \times B}{N - B}$.

On obtient P_c en échangeant les rôles de P_{Min} et P_{Max} :

$$P_c(n) = \frac{P_{Max} - P_{Min}}{N - B} \times n + \frac{P_{Min} \times N - P_{Max} \times B}{N - B} = P_{Max} + P_{Min} - P - a(n)\ .$$

Les deux fonctions d'appartenance étant disponibles, il ne reste plus qu'à développer le programme du robot.

On commence par poser certaines valeurs constantes :

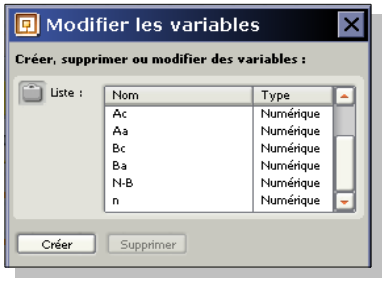
$P_{Min} = 25$

$P_{Max} = 75$

Noir = 16

Blanc = 50

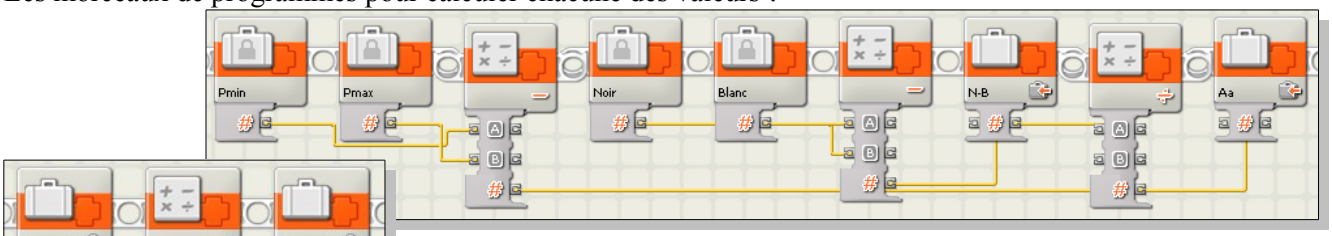
Et on va définir des variables utiles au programme. En effet, les fonctions d'appartenance ont des coefficients, calculés à partir des constantes précédentes ; lesquels coefficients sont considérés comme constants lors du fonctionnement du robot. Il n'est pas besoin de les recalculer dans chaque boucle, une seule fois suffit. Et ceci dans le but d'accélérer l'exécution du traitement du suivi de ligne, en évitant de refaire des calculs identiques.

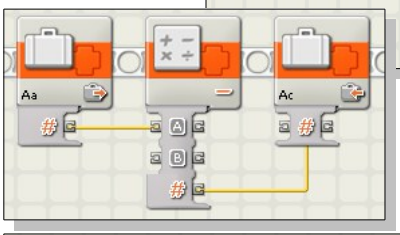


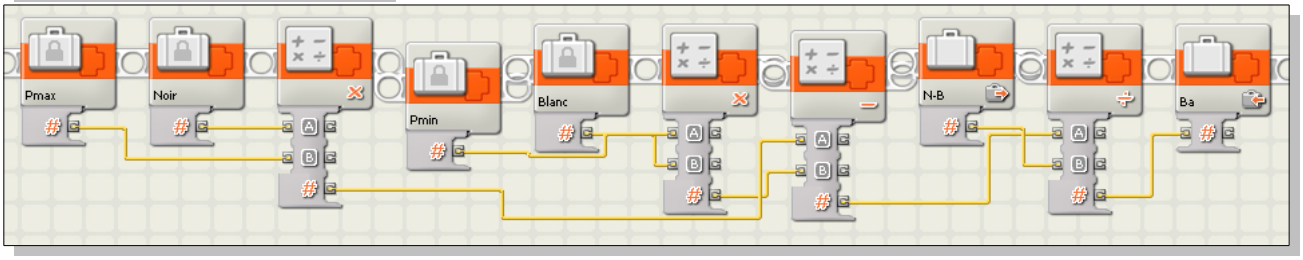
$$a_a = \frac{P_{Min} - P_{Max}}{N - B}\ ,\ a_c = \frac{P_{Max} - P_{Min}}{N - B} = -a_a$$

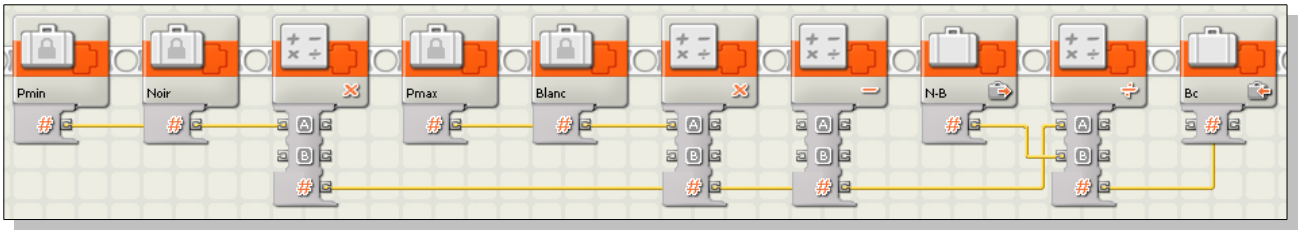
$$b_a = \frac{P_{Max} \times N - P_{Min} \times B}{N - B}\ ,\ b_c = \frac{P_{Min} \times N - P_{Max} \times B}{N - B}$$

Les morceaux de programmes pour calculer chacune des valeurs :



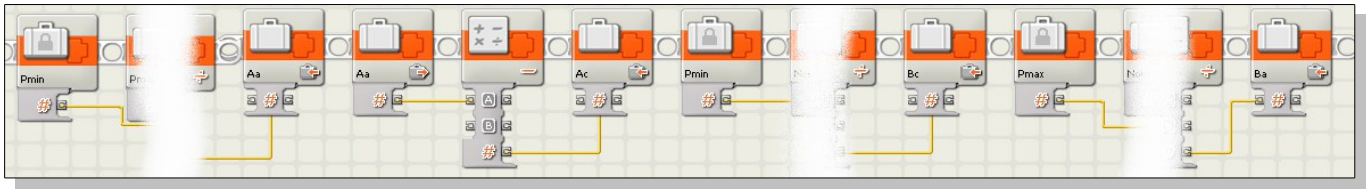






Pour des raisons de mise en page, chaque calcul de variable a été placé sur une ligne. Ces programmes, réalisés dans le logiciel Lego MindStorms, sont en fait sur une ligne d'exécution unique : le calcul de a_a est immédiatement suivi de celui de a_c, puis vient le module pour b_a et finalement celui de b_c :

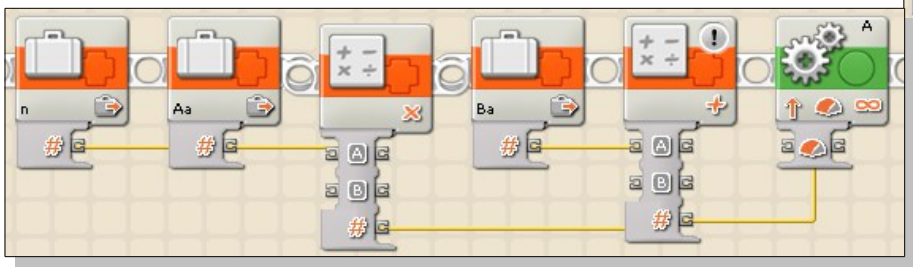
Le programme réel du suivi de ligne est alors abordé, et vient se placer à la suite de la partie précédente.



Le principe de base reste de lire la valeur de luminosité du capteur optique, ***n*** (ci-contre) :

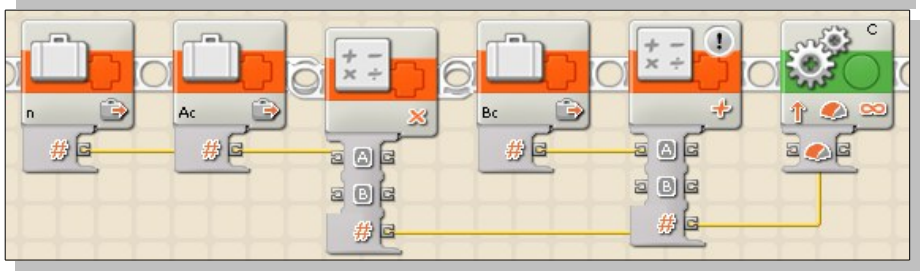
puis de calculer la puissance à appliquer sur chacun des moteurs, en utilisant les fonctions d'appartenance :

$P_a(n)$:

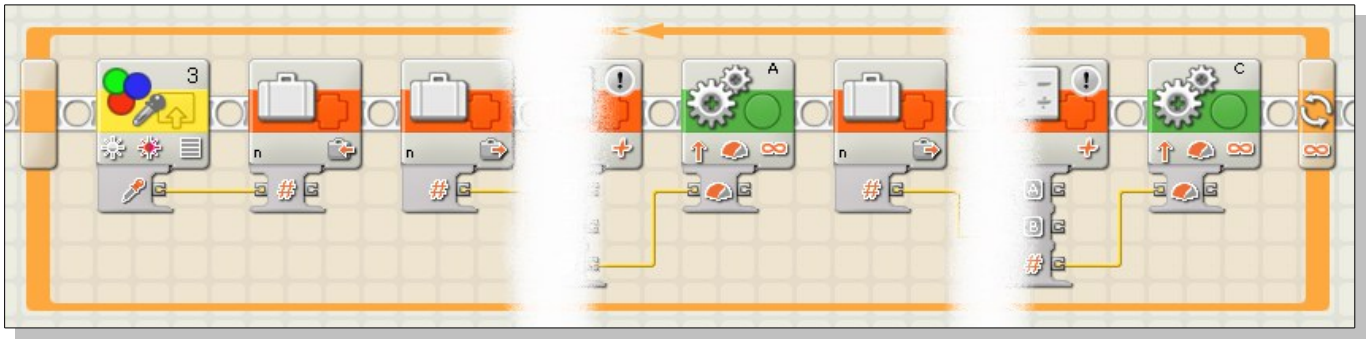


$P_c(N)$:

(on pourrait simplifier).



Ces opérations sont réalisées dans une boucle infinie :



Contre toute attente, ce programme, assez simple dans sa construction, ne fournira pas de bons résultats. Effectivement, les puissances calculées pour les moteurs doivent être appliquées simultanément sur les deux moteurs, pour que le robot suive correctement la ligne.

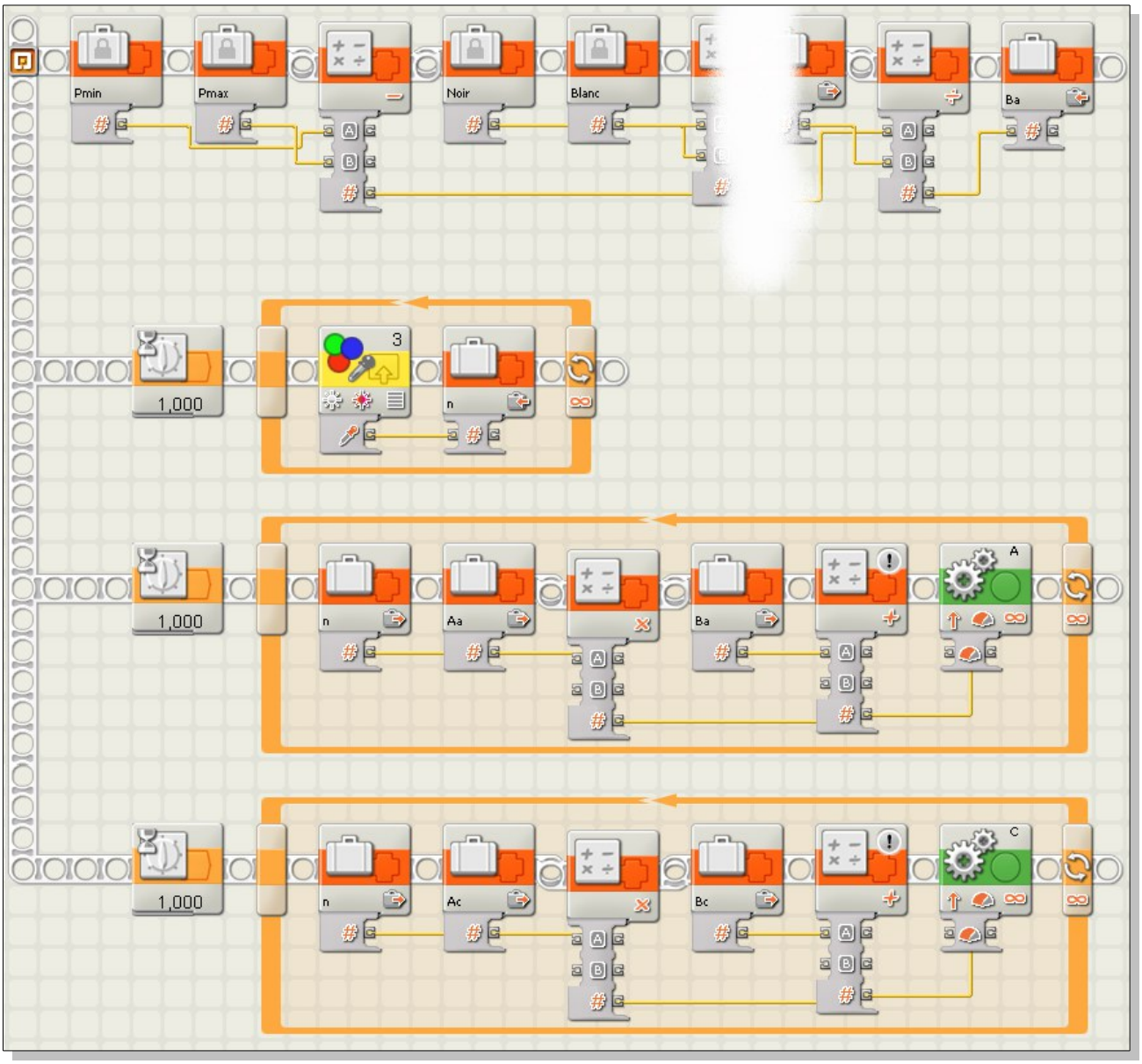
Cette constatation, observée expérimentalement par les élèves, va conduire l'enseignant à parler de multitâche : deux (ou plus) morceaux de programmes, qui doivent s'exécuter en même temps. Notez qu'ici, il n'est pas question de rentrer dans le détail de ce qu'est un programme multitâche, mais seulement de montrer que ce type de programmation peut résoudre des problèmes plus aisément.

Le logiciel Lego MindStorms offre la possibilité de programmer non seulement séquentiellement (les blocs sont sur une même ligne d'exécution horizontale : ils sont exécutés les uns après les autres) ; mais aussi parallèlement (plusieurs lignes d'exécutions séquentielles sont placées les unes sous les autres, les blocs d'une même ligne d'exécution se succèdent séquentiellement, mais toutes les lignes sont exécutées en même temps de façon concurrente).

Une des modifications du programme serait alors de produire 4 lignes d'exécutions différentes :

- la première permettant l'initialisation et le calcul des variables ;
- une deuxième récupère, en continu, la valeur de la luminosité sur le capteur et la place dans ***n*** ;
- les deux dernières sont chargées de calculer, en boucle, les puissances, au moyen des fonctions d'appartenance $P_a(n)$ et $P_c(n)$, et de les appliquer sur le moteur adéquat.

Le programme final est donc comme indiqué ci-dessous.

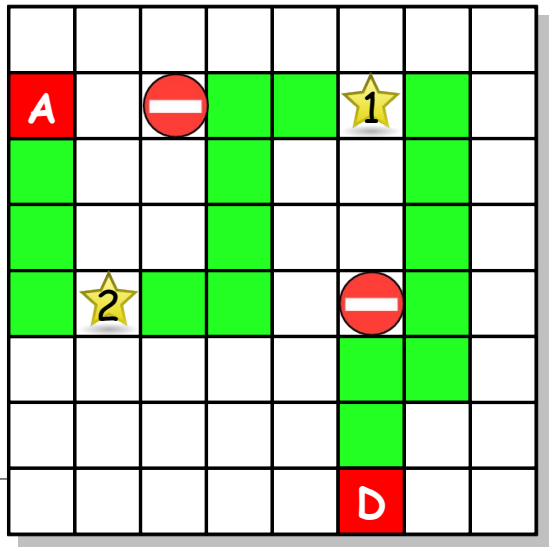


Notez qu'on a introduit une petite temporisation d'une seconde avant de commencer à réaliser les boucles, afin d'être sûr que la ligne d'exécution de l'initialisation des variables, s'est correctement déroulée, et est achevée.

Annexe 2 : mise en œuvre du projet

Pour cette partie, nous considérons un terrain de jeu réduit à 8x8 cases. Il n'est question ici que de montrer la faisabilité du projet demandé aux élèves.

1 / Programmer un parcours simple du robot.



En partant du point de départ **D**, le robot doit rallier les points jalons 1 puis 2, pour aboutir à la case d'arrivée **A**.

Des obstacles bien balisés sont placés sur le terrain.

Les élèves préparent le parcours sur papier, montré ici en vert. Le robot est simultanément piloté à distance par télécommande, pour permettre à ces mêmes élèves d'appréhender l'environnement et les difficultés potentielles.

Prenons par exemple le trajet indiqué sur le plan ci-dessus.

Un algorithme sommaire est alors écrit :

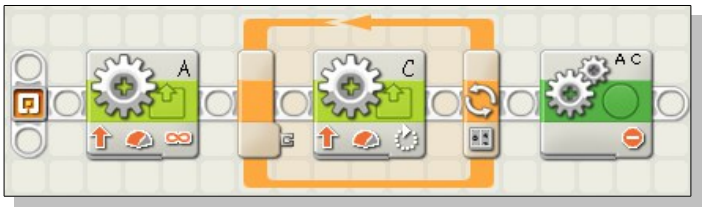
```
Avancer de deux cases
Pivoter vers la droite de 90°
Avancer d'une case
Pivoter vers la gauche de 90°
Avancer de quatre cases
Pivoter vers la gauche de 90°
Avancer d'une case
Réaliser l'action prévue sur les points jalons
Avancer de deux cases
Pivoter à gauche de 90°
Avancer de trois cases
Pivoter vers la droite de 90°
Avancer de deux cases
Réaliser l'action prévue sur les points jalons
Avancer d'une case
Pivoter vers la droite de 90°
Avancer de trois cases
```

Le problème se ramène à des actions primitives connues, qui ne devraient pas poser de problème aux élèves, à ce stade de l'activité.

En supposant que les cases soient de dimensions 20 cm x 20 cm, il est facile de faire le programme correspondant à chaque action primitive.

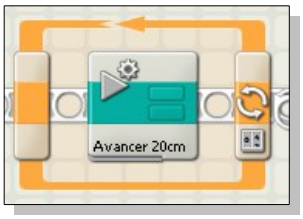
1.1 Avancer de 20 cm

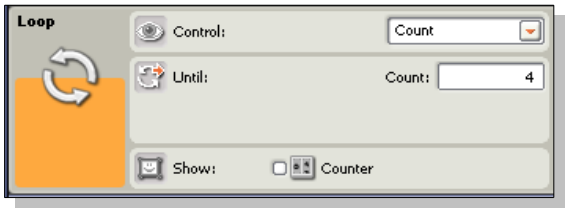
Les roues, fournies avec la version NXT 2.0, ont un diamètre de 4,32 cm. Un tour de moteur fait donc parcourir 13,57 cm au robot ($4{,}32\,\text{cm} \times \pi = 13{,}57\,\text{cm}$). Pour avancer de 20 cm, les roues doivent effectuer 1,47 tours, soit 530,58 degrés. Le morceau de programme est alors :



Nous pouvons à cette occasion définir un Bloc *Personnel* (voir le menu Édition/Créer un nouveau Bloc…). Ce bloc permet de faire avancer le robot de 20 cm, soit une case, et ressemble à :

Et pour faire avancer le robot de plusieurs cases, quatre par exemple, il suffit d'exécuter le bloc *Avancer 20 cm* dans une boucle de comptage :





L'expérimentation montre qu'une certaine dérive, à droite ou à gauche, se produit sur des ligne droites un peu trop longues. Ceci est lié au fait que les moteurs, ayant pourtant les mêmes puissances appliquées, ne tournent pas exactement à la même vitesse : une différence, même infime, est parfaitement visible dans la trajectoire. On veillera à éviter de trop longues lignes droites.

1.2 Pivoter à droite, à gauche de 90°

Le principe a déjà été abordé : on réutilise les résultats trouvés dans l'Annexe 1. Les morceaux de programme sont donc :

pour pivoter à droite

pour pivoter à gauche

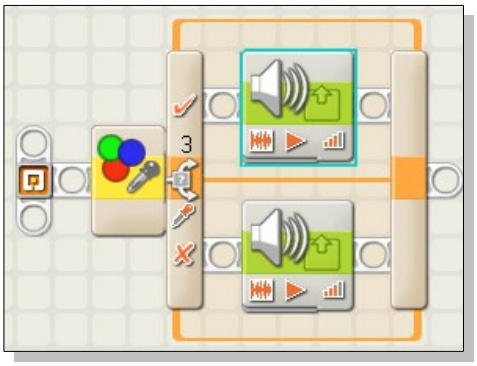
Ici aussi, une erreur de précision est très probable : la valeur de consigne en rotation du moteur, étant exprimée uniquement par une valeur entière, le robot tournera soit un peu moins soit un peu plus que les 90° requis. La trajectoire rectiligne suivante n'est donc plus parfaitement dans l'axe prévu ; nous en reparlerons plus tard.

1.3 Réaliser une action

N'importe quelle action est possible sur les points jalons. Il serait pourtant judicieux de rester dans l'ordre du raisonnable, afin que le projet puisse aboutir. Par exemple, faire faire un tour complet au robot sur lui-même, serait une source d'erreur de placement supplémentaire, pouvant fortement perturber les élèves.

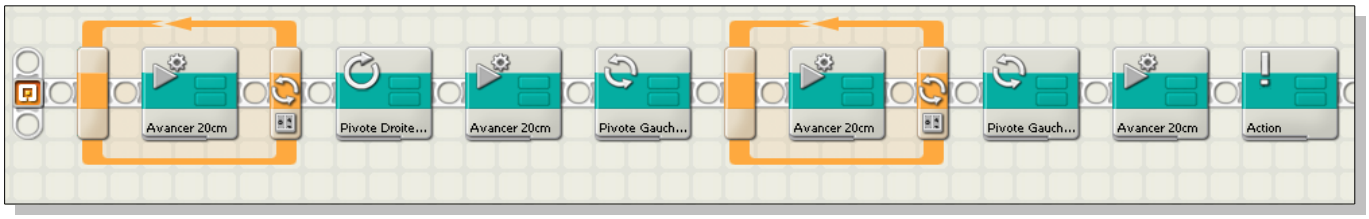
J'ai choisi, pour l'action au point jalon, de faire détecter la couleur de la case, afin de simuler la découverte ou non d'une météorite. Si la case est noire, le robot signale une découverte en émettant le son standard « Detect ». Le son « Malfonction » est joué dans tout autre cas.

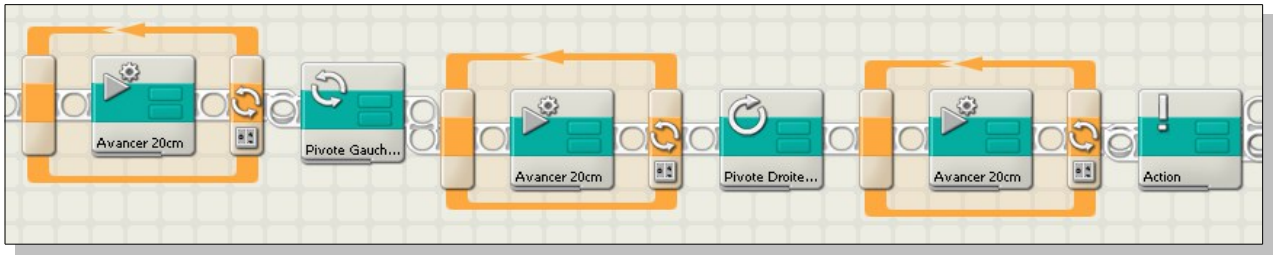
Le programme et le bloc *personnel* de l'action sont alors :

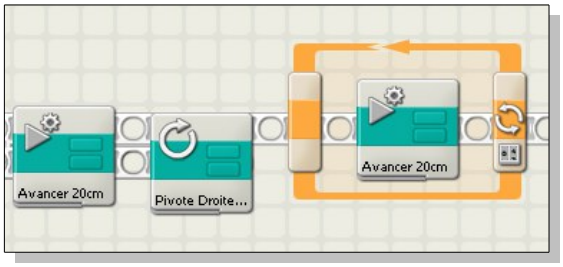


1.4 Le programme complet peut maintenant être réalisé

On utilise les blocs personnels créés :







Lors de l'exécution, on peut remarquer que le robot dévie rapidement, après avoir effectué un virage de 90° : sa nouvelle trajectoire n'est pas exactement perpendiculaire à celle avant de tourner !

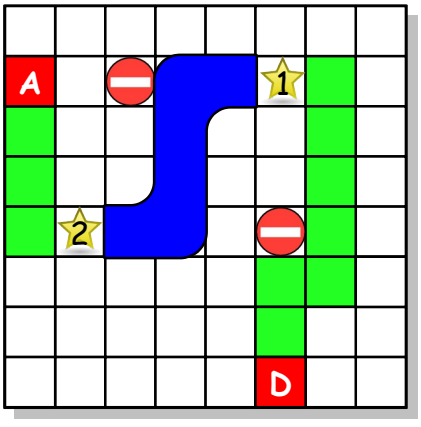
Une des causes en est que le bloc *Moteur* est activé durant le temps nécessaire pour tourner d'un certain nombre de degrés. Les calculs effectués dans l'annexe 1 nous indiquent que 264,58° sont requis pour pivoter de 90° exactement. Seulement, ce même bloc *moteur* n'accepte que des valeurs entières de consigne : nous devons alors choisir de l'arrondir, soit à 264° (perte de β=0,58° de tour de roue), soit à 265° (gain de β=0,42° de tour de roue).

En reprenant les formules précédentes : $\alpha = \frac{\beta \times 2 \times r}{e}$.

Perte de 0,58° de tour de roue : $\alpha = \frac{0{,}58 \times 2 \times 2{,}16}{12{,}7} = 0{,}1972$ le robot tourne de $90 - 0{,}1972 = 89{,}8027°$.

Gain de 0,42° de tour de roue : $\alpha = \frac{0{,}42 \times 2 \times 2{,}16}{12{,}7} = 0{,}1428$ le robot tourne de $90 + 0{,}1428 = 90{,}1428°$.

De surcroît, plus le nombre de virages dans un même sens est important, plus les erreurs d'arrondi se cumulent, faisant dévier d'autant plus rapidement le robot.



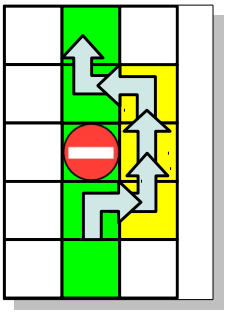
Il n'est pas facile d'apporter une solution à ce problème ; car plus le robot s'éloigne du centre du pivot, plus il dévie, et plus il faudra corriger sa trajectoire. À ce niveau-là, le programme devient réellement complexe. Une solution simple pour rattraper ces erreurs d'arrondi, peut être de faire suivre au robot une ligne de couleur contrastée sur le terrain. Le robot est remis dans une bonne direction à un certain moment du parcours. L'entrée de la ligne à suivre sera prévue en conséquence, et devra être assez large pour que le robot déviant entre dans la zone sombre, et accroche la bordure de la bande à suivre.

Pour faciliter les choses, une bande bleue, de la largeur d'une case complète, pourrait être tracée afin que le robot sache qu'il doit suivre une ligne dans cette zone. Le terrain ressemblerait à la figure ci-contre.

1.5 Éviter un obstacle imprévu

Sur la base du programme précédent, on peut ajouter la détection et le contournement des obstacles imprévus. Le capteur ultrasonique est mis à contribution pour cela. Le robot avance en ligne droite, selon son programme, tant que rien n'empêche sa progression. Lorsqu'un obstacle est détecté, une branche conditionnelle gère l'évitement.

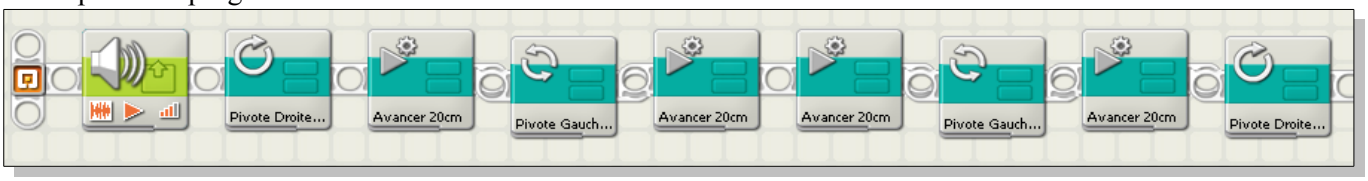
Son algorithme est :



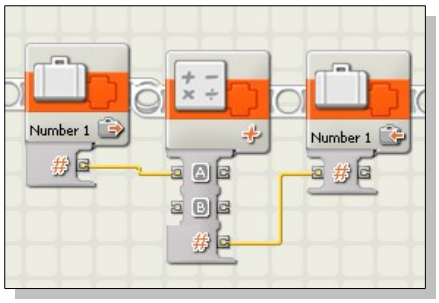
```
Emettre le son « Object detected »
Pivoter de 90° vers la droite
Avancer d'une case
Pivoter de 90° vers la gauche
Avancer de deux cases
Pivoter de 90° vers la gauche
Avancer d'une case
Pivoter de 90° vers la droite
```

Au final, et sans tenir compte des erreurs liées aux virages évoquées plus haut, le robot a retrouvé sa trajectoire initiale, comme s'il n'avait avancé que de deux cases en ligne droite.

Cette partie du programme est :

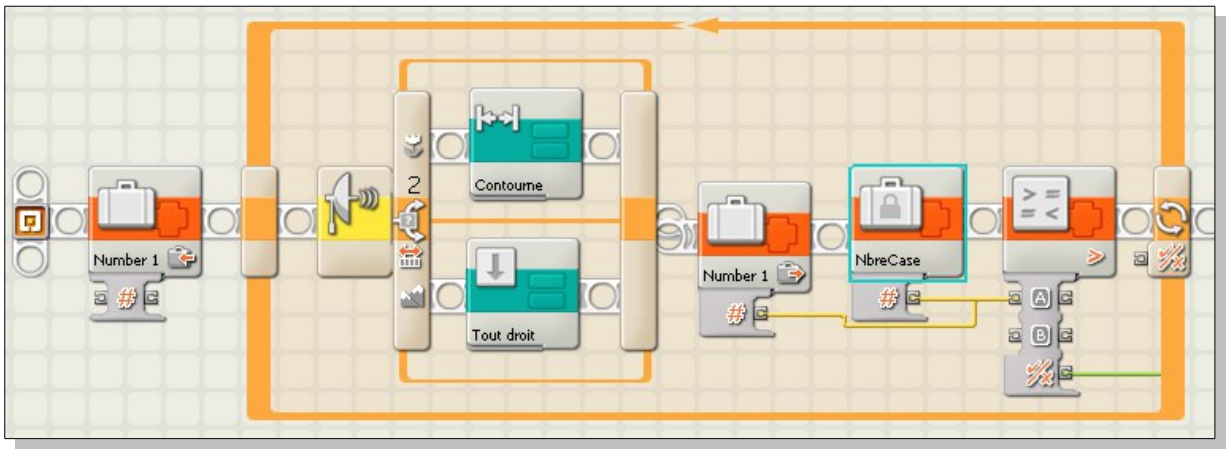


Nous allons voir qu'il est nécessaire de mémoriser le fait que le robot ait parcouru l'équivalent de deux cases en trajectoire rectiligne. Pour ceci, le programme se termine par l'ajout de ces deux cases au nombre de cases parcourues.



Ce module fait l'objet d'un nouveau bloc *personnel :*

Le programme de détection de l'obstacle final :



Ce programme peut avantageusement remplacer le programme du bloc personnel *Avancer de 20cm* réalisé un peu avant.

Comme nous le voyons, d'un point de vue avance du robot, la branche de la conditionnelle exécutée lorsqu'aucun obstacle n'est détecté (en bas sur le programme), permet un parcours de 20 cm, tandis que son antagoniste (en haut) fera un trajet équivalent à deux cases, soit 40 cm. Cette différence doit être prise en compte lorsqu'on doit faire avancer le robot d'un nombre de cases (NbreCase dans le programme) précis. La variable Number1 est incrémentée de 1 dans le bloc personnel *Tout droit* et de 2 dans le bloc *Contourne.*

Notons aussi que les obstacles imprévus ne sont ici pris en compte, que lors de l'avance du robot, et non durant les virages, ni l'éventuel suivi de ligne.