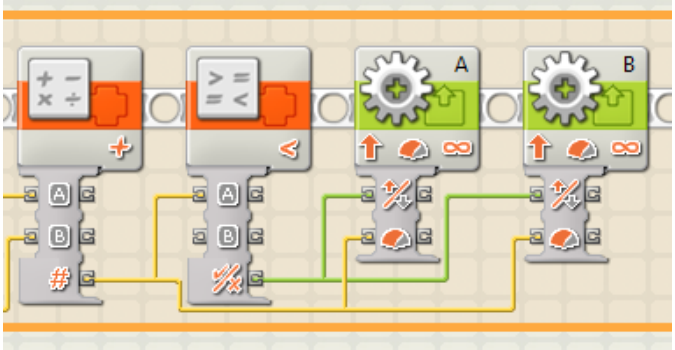
Construire et Programmer son RobotSegwayNXT

Comment réaliser un robot qui imite le fonctionnement d'un Segway ?

TPE d'Axel Lovera et Alistair Laurière

SOMMAIRE :



INTRODUCTION :

Tu as pour but de Réaliser un robot qui imite le fonctionnement d'un segway, c'est-à-dire un robot auto-balancé sur deux roues. Tu possèdes un kit lego Mindstorm NXT et tu veux aller plus loin dans la construction et la programmation de ton robot.

Notre livret va t'orienter et t'aider sur les bases pour construire un robot NXT plus avancé. Si tu as déjà construit des robots exécutants diverses actions et programmes basiques fournit par le kit et site officiel, notre tutorial te sera plus accessible.

Le robot que nous te proposons est un robot équilibriste s'inspirant des Segways.

Pourquoi ce modèle ? Comme dans la vraie vie, ces engins tenant en équilibre fascinent, ils repoussent les codes des véhicules. Ce modèle est hors du commun et très surprenant.

Tous les robots NXT ayant pour fonction de se déplacer possèdent quatre ou trois roues comme des voitures, pour avoir trois points d'appuis minimum. Les motos ou vélos tiennent en équilibre par la dynamique des cycles, leur trajectoire est à sens unique, les forces qui s'appliquent sur eux comme la gravité peuvent les faire tomber. Par l'action du cycliste de tenir en équilibre, donc de maintenir son centre de gravité, et par la vitesse, les forces centrifuges s'exerçants sur lui se compensent. C'est ce qui se passe lorsque tu fais tourner une toupie ou que tu fais du vélo. Le reproduire avec un Mindstorm est encore plus compliqué qu'un segway.

C'est un japonais qui l'a réalisé. Il n'a pas reproduit le sens d'équilibre d'une personne, à savoir un repositionnement du cycliste basé sur un capteur gyroscopique, mais en installant un robot pédalant et gardant son équilibre par la direction de son guidon. Il tourne à droite lorsqu'il tombe vers la droite pour se remettre droit et idem pour la gauche mais le faire aller parfaitement droit n'est donc pas possible. Il sait aussi tourner en prenant en compte l'inclinaison pour effectuer un virage.

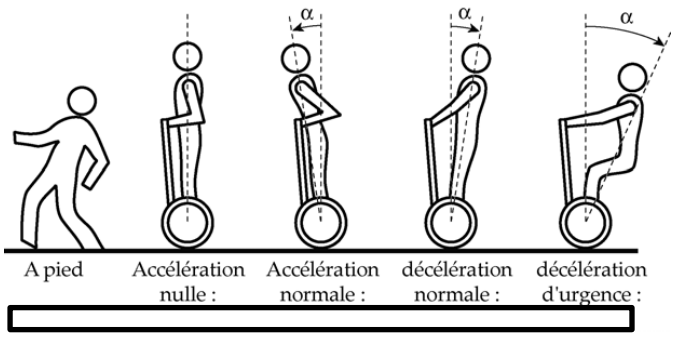
Le principe du Segway est plus simple, la fonction de maintien en équilibre ne relève que du fait de ne pas basculer en avant ou en arrière, il ne peut pas tomber sur les coté car ses deux pneus larges de chaque coté le maintiennent.

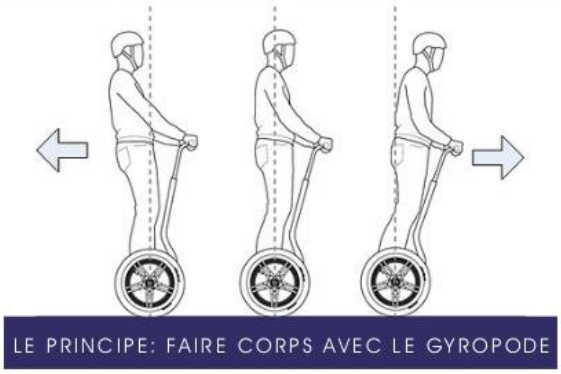
Ton objectif est donc de le programmer pour le maintenir droit comme un Segway et c'est là le challenge.

Les Segways :

Depuis quelques années, leur popularité ne fait qu'augmenter, désormais ils sont partout, surtout en Amérique et sur Internet. Mais après avoir été impressionné par les rares personnes chanceuses qui le conduisent, on se demande comment ça marche et quel est sa réelle utilité pour les secteurs professionnels comme la sécurité civile. Ils prennent moins de place qu'une voiture et permette de se déplacer à des vitesses supérieures à 20km/h. Ils remplacent principalement les surveillances à pied, à vélo et à cheval, pour toutes ces raisons, c'est le juste milieu entre pratique, encombrement et déplacement.

Les Segways fonctionnent avec un capteur gyroscopique qui va prendre en compte l'inclinaison du véhicule et ne va bouger pour se maintenir qu'après un certain point car la personne le conduisant peut se tenir dans un champ de degrés différent. Son système pour se remettre à un angle de 90° : quand la personne va pousser le guidon ou manche vers l'avant, les moteurs avancent. L'orientation est contrôlée par le manche, pour tourner à droite par exemple, la roue gauche va tourner plus vite. La vitesse supérieure de la roue gauche à celle de droite est proportionnelle à l'angle du virage.





Il existe depuis quelques temps des modèles réduit de Segways appelés « hoverboards » qui reflètent plus les raisons de son utilité générale. Ces modèles réduits plus abordables prennent autant de place que nos pas, tout en se déplaçant à 10km/h max. Ils fonctionnent différemment, la même chose de chaque côté pour individualiser les vitesses des roues, un capteur gyroscopique captant une inclinaison. Penchez vers l'avant et le moteur fait avancer les roues. La vitesse est proportionnelle à l'inclinaison et l'inclinaison de chaque côté remplace la fonction du guidon des Segways.

Le Segway est donc un gyropode, c'est-à-dire un véhicule électrique monoplace, constitué d'une plateforme munie de deux roues sur laquelle l'utilisateur se tient debout, d'un système de stabilisation gyroscopique et d'un manche de maintien et de conduite.

Lorsque Dean Kamen présenta pour la première fois son Segway Personal Transporter (PT) lors d'une émission "Good Morning America" de la chaîne ABC, il le décrivit comme "le premier moyen de transport auto-balancé pour humains". En observant le véhicule en action, on comprend instantanément ce qu'il voulait dire. Au contraire d'une voiture, le Segway n'a que deux roues. Mais, et c'est là sa caractéristique, il est capable de se maintenir en position verticale sans assistance.

La Construction

Mais alors comment construire un RobotSegwayNXT ?

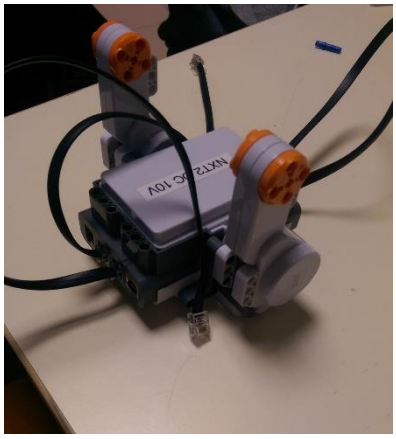
Tout d'abord il faut s'assurer d'être munis d'une boite complète de Lego Mindstorms NXT. Ensuite, la construction est, elle, basée sur le model réel d'un segway.

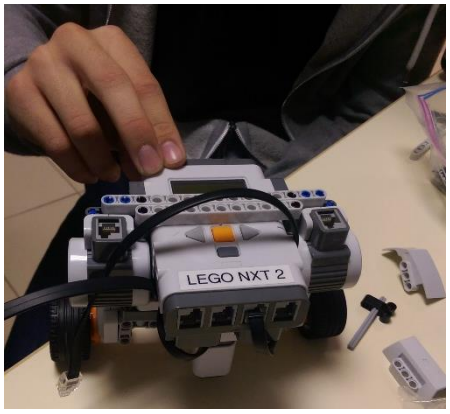
Tu as vu comment une personne se tient sur un segway, nous allons donc construire le robot comme un humain conduisant un segway, les raisons de cette structure sont purement esthétiques.

Le robot se présente en deux parties, l'humain par un moteur et d'autres pièces puis un capteur ultrasons dessus semblable au buste une tête et des bras, se reliant par un guidon.

Le Segway est remplacé par la brique comme base et les deux moteurs et leur roue. Le capteur lumineux est placé en dessous de la brique remplacent les systèmes gyroscopiques à l'intérieur d'un vrai segway.

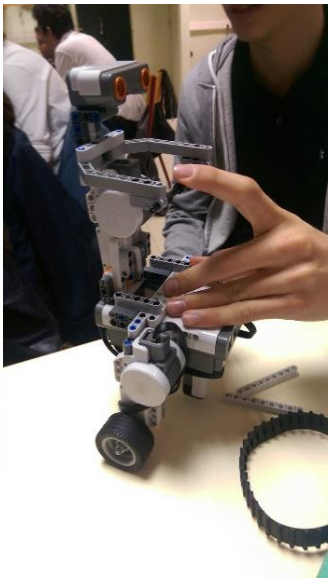
Pour la partie du bas, le châssis du segway, on raccorde à la brique NXT deux moteurs et roues de chaque côté au milieu de l'écran, le poids est réparti car la batterie pousse la brique vers l'avant mais le conducteur placé par-dessus rééquilibre la structure.

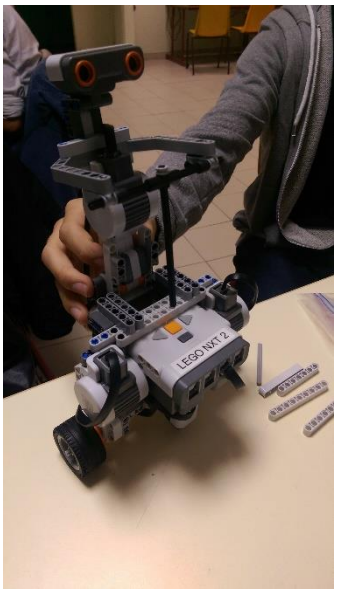




Par la suite, on crée le corps de la personne et on gère le nombre de pièces qui la constitue afin de gérer son poids.

Enfin on assemble les deux parties et on modifie le « cable management » et on gère encore la répartition du poids pour permettre le bon fonctionnement du robot.





Pour une explication étape par étape plus approfondie, guidée, du style manuel lego, tu peux aller chercher des autres modèles sur internet mais avec des centres de gravité plus élevés ou semblable : http://www.nxtprograms.com/NXT2/segway/steps.html

Le robot peut aussi être monté avec un capteur gyroscopique au lieu d'un capteur d'ultrason, comme ceci :

Cela ne change rien au robot, ça ne sert que pour l'équilibre pour le moment. Le capteur qui va nous intéresser et qui va être utilisé pour la programmation prochainement est le capteur lumineux situé ici:

La Programmation

Maintenant vous passez au plus compliqué, la programmation du RobotSegwayNXT.

Il faut d'abord se munir d'un logiciel LEGO MINDSTORMS NXT 2.0 indispensable pour la programmation, (Logiciel que vous pouvez installer grâce au CD fourni dans la boite) et commencer à vous familiariser avec ce logiciel, c'est grandement recommandé.

L'explication de la programmation du RobotSegwayNXT va s'effectuer progressivement sur plusieurs tests/programmes pour vous faire comprendre le fonctionnement en détails de cette programmation et comment faire pour arriver à ce que le robot tiennent en « équilibre » tout seul sur ses deux roues.

L'originalité de ce RobotSegwayNXT c'est qu'il est programmé avec un capteur lumineux qui va faire en sorte de respecter un seuil de luminosité donné qu'il doit suivre pour rester à la verticale et donc en équilibre dans cette position. La position du robot faisant en effet modifier la distance Robot-Sol…la lumière détectée (après émission par le capteur lui-même sous le robot) va varier si le robot bouge.

On veut que le RobotSegwayNXT soit auto-balancé comme un segway, c'est-à-dire qu'il tienne en équilibre. Mais alors comment faut-il faire ?

On sait que lorsque l'on place le robot droit il tombe soit en avant, soit en arrière. Il serait donc logique que lorsque qu'il :

-tombe en avant les moteurs doivent faire tourner les roues en avant pour rattraper la chute et se rééquilibrer et donc se replacer à la verticale.

-tombe en arrière les moteurs doivent faire tourner les roues en arrières pour les mêmes raisons.

Aussi vous allez pouvoir constater, en testant le capteur lumineux, que lorsque le robot tombe en avant le capteur identifie une luminosité reflétée supérieure que lorsqu'il tombe en arrière. Valeur de la luminosité pouvant aller de 0 à 100

-Valeur de la luminosité à la verticale X

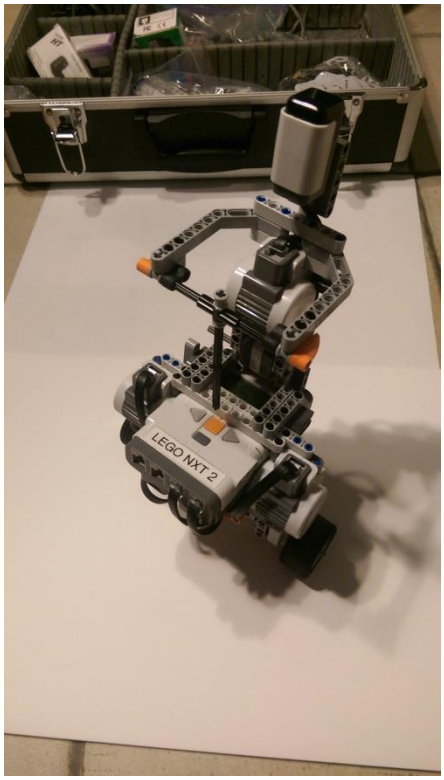
-Valeur de la luminosité quand le robot tombe en avant Xavant>X

-Valeur de la luminosité quand le robot tombe en arrière Xarrière<X

La puissance des moteurs du robot pouvant aller de 0 à 100.

ATTENTION *Avant le démarrage des programmes sur le robot, il faut vérifier à ce qu'il démarre verticalement et faire en sortes de rester dans un endroit où la luminosité de la pièce reste la même pendant le déroulement du programme sinon le seuil pourrait changer. Il faut aussi faire en sorte de pouvoir mettre en marche le robot sur une surface totalement plane. Pour la mise en marche du robot voir la feuille du fonctionnement du robot.*

RobotSegwayNXT



Première partie : Découverte

Test 1 :

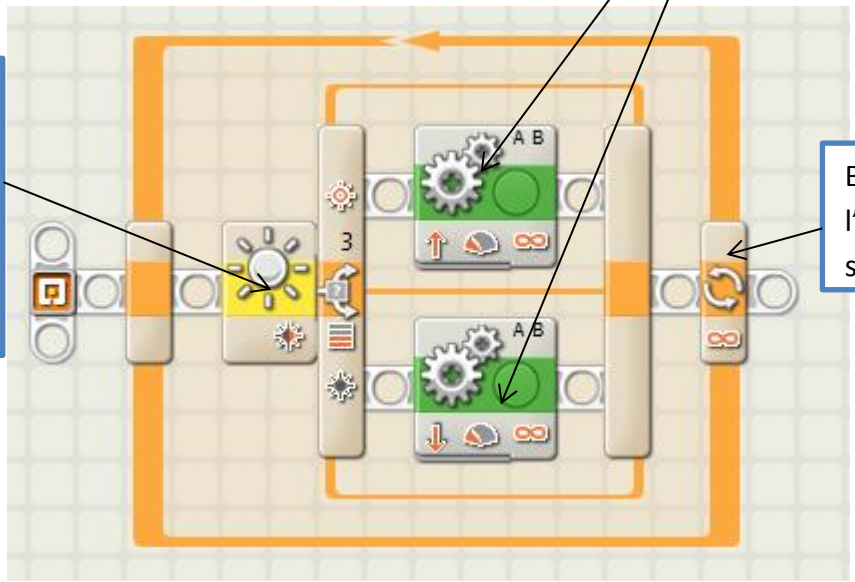
Principe :

Lorsque le robot constate, avec le capteur lumineux, qu'il tombe en avant ses moteurs font tourner les roues en avant et vice-versa.

Programme :

Bloc moteur : donnant la direction, la puissance et la durée des moteurs A et B

Capteur lumineux sur port 3 de la boite NXT: réglé avec un seuil de luminosité donné à respecter



Boucle infini : l'action à l'intérieur se répète à l'infini

Explication du programme :

Si la valeur lumineuse détectée est supérieure à celle qui doit être respectée alors le robot avance avec une puissance donnée. Par contre si la valeur lumineuse détectée est inférieure à celle qui doit être respectée alors le robot recule avec une puissance donnée.

Observation du robot en marche :

On va pouvoir constater que le robot réagit différemment en fonction de la puissance que l'on donne à ses moteurs. En effet quand la puissance est à 50, le robot réagit bien mais les moteurs ne sont pas assez puissants et donc il ne tient pas en équilibre et tombe.

-A 75 le robot est trop brusque et ne tient pas du tout en équilibre et tombe rapidement.

-A 100 bien sur le robot réagit avec une puissance encore plus forte et donc dès qu'il bascule un peu d'un côté ou de autre les moteurs le font donc aller dans le sens contraire à une vitesse tellement grande qu'il tombe directement après.

-A 25 le robot dans ce cas ne réagit pas vite et donc dès qu'il bascule un peu la puissance des moteurs n'est pas assez forte pour le rattraper.

Voir clé USB pour les vidéos tests.

Test 2 :

Ce qui nous amène au deuxième test. On voit bien que la puissance des moteurs du robot doit varier en fonction du basculement du robot. Si le robot tombe doucement en avant ou en arrière la puissance ne doit pas la même qui s'il tombe rapidement vers le sol, là la puissance devra être plus forte qu'auparavant.

Principe :

On pourrait donc essayer de faire varier la puissance des moteurs du robot en fonction de la luminosité perçue par le capteur lumineux. Imaginons le seuil de luminosité à respecter pour que le robot soit droit égal à X.

Par exemple : Quand le capteur lumineux voit une luminosité présente dans l'intervalle [X+1 ; X+5], c'est-à-dire que le robot est en train de tomber en avant car la luminosité augmente un peu, on donne une puissance moyenne. Mais si la luminosité est dans l'intervalle [X+6 ; X+10], ce qui signifie ici que le robot tomberait plus en avant et donc la puissance devra être plus forte.

Et c'est la même chose quand le robot tombe en arrière. Si il ne bascule que un petit peu, la luminosité sera dans l'intervalle [X-5 ; X-1] et la puissance des moteurs devra être plus faible qui si la luminosité est présente dans l'intervalle [X-10 ; X-6], ce qui signifie que le robot tombe plus vite en arrière et plus proche du sol, là la puissance des moteur devra être plus forte.

Programme avec explications:

Le programme va être composé de quatre possibilités d'intervalle à respecter suivant le principe expliqué juste avant. Les deux premières boucles vont servir à faire fonctionner le robot quand il tombe en arrière en fonction de la luminosité et les deux dernières vont faire fonctionner le robot quand il tombe en avant en fonction de la valeur de sa luminosité aussi.

Prenons pour exemple un cas où le seuil de luminosité à respecter est 54 :

-On décide deux intervalle de luminosité perçu par le capteur quand il tombe en arrière :
[49 ; 53] soit un peu en arrière et [45 ; 48] un peu plus encore

-On décide deux intervalle de luminosité perçu par le capteur quand il tombe en avant :
[55 ; 58] soit un peu en avant et [59 ; 62] un peu plus encore

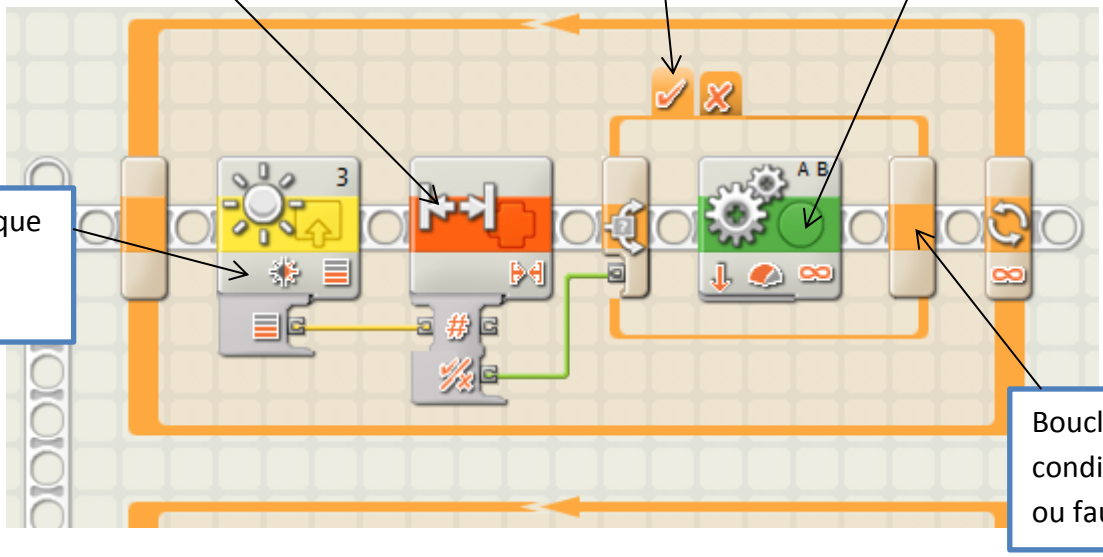
Pour le premier intervalle [49 ; 53] quand le robot tombe en arrière (on rappelle que dans ce cas les moteurs doivent faire aller le robot en arrière pour rattraper)

La valeur de la luminosité mesurée est comparée l'intervalle [49 ; 53]

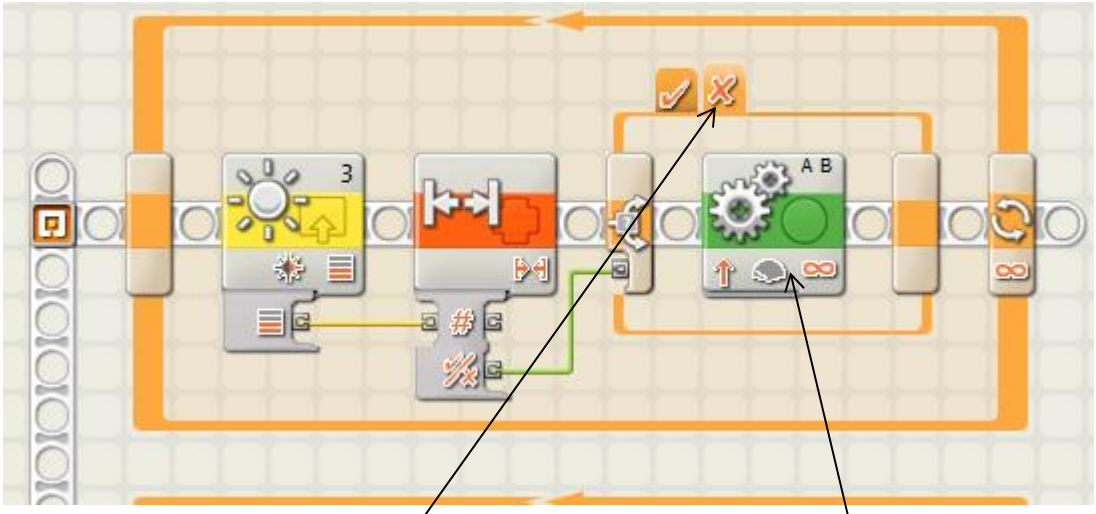
Si elle est comprise dans l'intervalle [49 ; 53]

Alors les moteurs A et B vont fonctionner pour aller en arrière à une puissance de 80 par exemple

Luminosité que le capteur perçoit



Boucle de condition (vraie ou fausse)



Si la luminosité mesurée par contre n'est pas comprise dans l'intervalle [49 ; 53]

Alors les moteurs A et B vont fonctionner vers l'avant à une très faible puissance comme 5 qui ne change rien appart pour ne pas bloquer les moteurs complètement.

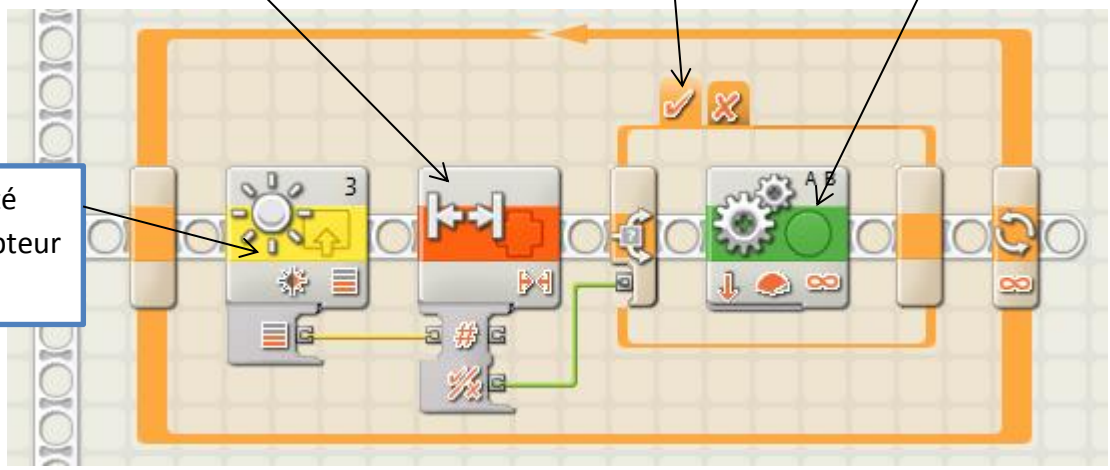
Pour le deuxième intervalle [45 ; 48] quand le robot tombe un peu plus en arrière :

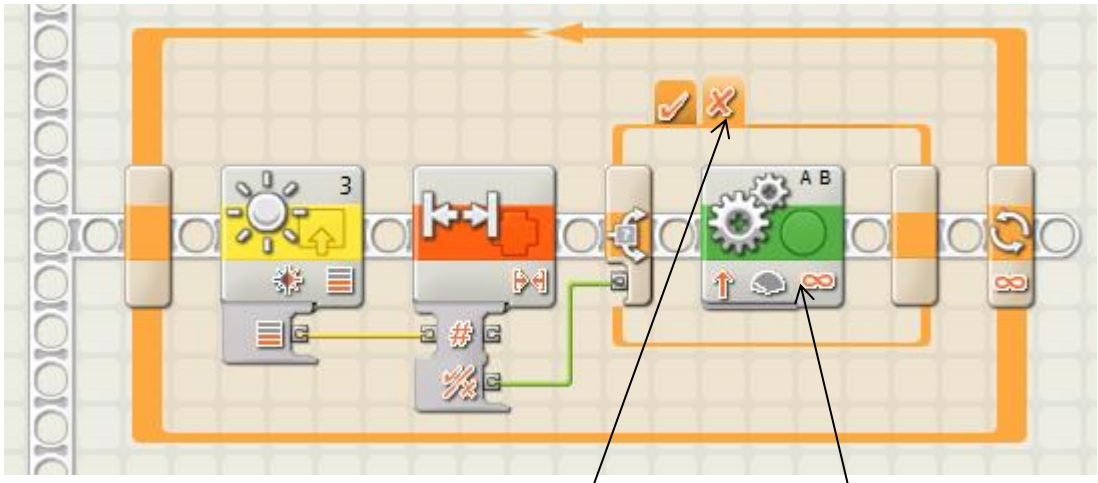
La valeur de la luminosité mesurée est comparée à l'intervalle [45 ; 48]

Si elle est comprise dans l'intervalle [45 ; 48]

Alors les moteurs A et B vont fonctionner pour aller en arrière à une puissance de 100 cette fois car le robot tombe plus bas ici

Luminosité que le capteur perçoit





Si la luminosité mesurée par contre n'est pas comprise dans l'intervalle [45 ; 48]

Alors les moteurs A et B vont fonctionner vers l'avant à une très faible puissance comme 5 (pour les mêmes raisons que tout a l'heure)

Pour les deux autres intervalles [55 ; 58] et [59 ; 62] c'est-à-dire quand le robot tombe vers l'avant il suffit juste de faire la même chose pour chaque intervalle mais en inversant le sens des moteurs car le robot tombant en avant les moteurs doivent aller en avant aussi.

Aussi il faut placer les boucles en parallèle pour qu'elles soient exécutées en même temps.

La difficulté ici est d'arriver à bien régler les intervalles en fonction de l'endroit où l'on se trouve, et aussi d'en trouver qui soit bien sûr possible à respecter. Il faut donc être bien sûr d'avoir relié tous les blocks nécessaires, ensembles et au bon endroit, et bien placer les boucles. Tout le problème ici est donc la précision mais rien n'est impossible.

Observation du robot en marche :

Il ne sert à rien de faire plus d'intervalles car quand le robot tombe, il n'a aucune chance de se rattraper quand il se trouve, ici, à une valeur de luminosité inférieure à 45 ou supérieure à 62.

En résumé, on va pouvoir constater que le robot réagit un peu mieux avec cette méthode mais il est toujours aussi brutal et ne tient pas en équilibre. Même si l'on baisse la puissance des intervalles le résultat reste le même : le robot ne tiendra pas. Il n'y aura pas dans ce cas assez de puissance pour rattraper le robot.

Voir clé USB pour les vidéos tests.

Deuxième partie : Contrôle par PID

On pourrait penser à assigner à chaque valeur de la luminosité une puissance précise pour les moteurs mais se serait trop long à faire pour savoir quelle puissance pour quelle valeur. Idem pour la programmation.

Ce qui nous amène maintenant à l'idée d'asservissement par PID en robotique. Le contrôle par PID (pour « proportionnel, intégral et dérivé ») est une méthode de régulation souvent employée pour les asservissements. L'asservissement c'est un système, capable d'atteindre et de maintenir une consigne grâce aux mesures qu'il effectue.

Pour faire simple, si on prend l'exemple d'un balai qu'on essaye de faire tenir à l'envers et en équilibre sur notre main. Faire de grands mouvements avec une vitesse quelconque ne suffira pas à faire tenir le balai en équilibre. Il faut faire plein de petits à-coups, non-brusques, variant en fonction de la vitesse de retombée du balai et anticipant le prochain mouvement pour faire tenir le balai droit presque parfaitement. Tout le problème de l'asservissement est là et le contrôle par PID est un moyen de le résoudre.

La relation du contrôle par PID et du type : e = étant l'erreur mesurée

$f(e) = Kp * e + Ki * \int edt + Kd * de/dt$

Plan à suivre: Il faut d'abord commencer par voir si un asservissement correct peut être obtenu ou approché en n'utilisant que le coefficient de proportionnalité (Kp). Ce n'est pas impossible, mais le robot sera instable (la moindre perturbation fera tomber le robot).

Puis il faut introduire l'intégrale ($Ki * \int edt$) dont la vertue est de faire converger plus rapidement l'asservissement en évitant les effets d'instabilité autour de la position cible.

Ajoutez ensuite la dérivée ($Kd * de/dt$), qui permettra de donner le "coup de boost" pour rattraper rapidement un écart avant qu'il ne devienne irrécupérable.

Il y a cependant l'inconvénient de dégrader la stabilité. En fait toute la difficulté des asservissements PID est de trouver le bon compromis.

Test 1 :

Ce premier test ne va utiliser que la partie P (Proportionnel) de PID, comme il en est question dans le plan.

Principe :

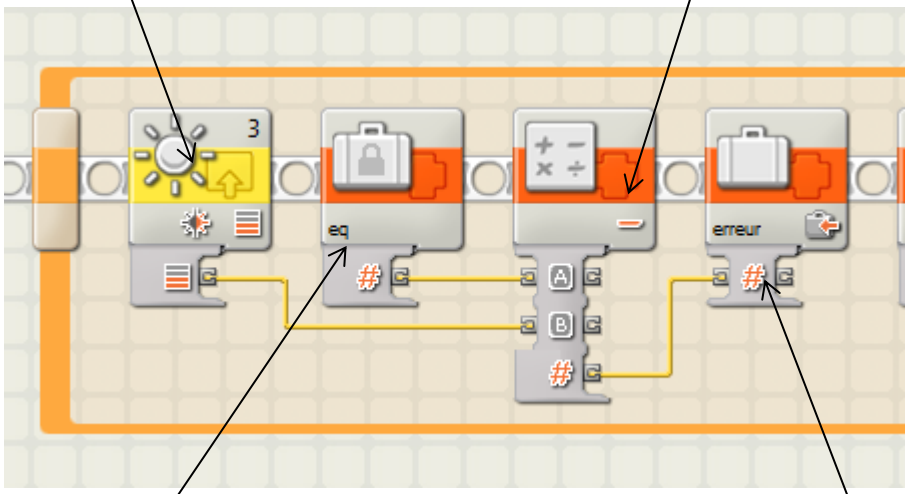
Dans le cas d'un contrôle proportionnel, l'erreur est virtuellement amplifiée d'un certain gain constant qu'il conviendra de déterminer en fonction du système. L'idée étant d'augmenter l'effet de l'erreur sur le système afin que celui-ci réagisse plus rapidement aux changements de consignes. Plus la valeur de kp est grande, plus la réponse l'est aussi. En revanche, la stabilité du système s'en trouve détériorée et dans le cas d'un kp démesuré le système peut même diverger.

Programme :

Il faut tout d'abord établir la différence entre le seuil de luminosité à respecter et celle mesurée pour assigner l'erreur

Capteur lumineux mesurant la valeur de la luminosité

On va soustraire à cette constante la valeur de la luminosité mesurée



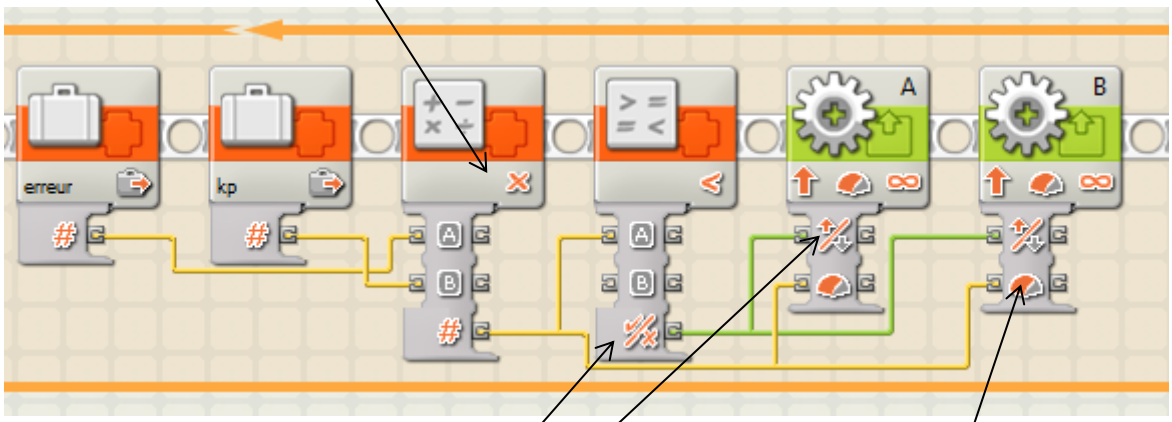
La constante à respecter pour que le robot soit droit, donc si on reprend l'exemple de la première partie, la constante à respecter ici est 54 qui est ici nommée « eq »

Le résultat sera donc l'erreur, c'est-à-dire le défaut de luminosité. Rappel : Si le robot tombe en avant la valeur de la luminosité sera supérieure au seuil et donc le résultat sera négatif, à l'inverse si la valeur mesurée et inférieure, le résultat sera positif.

Ensuite dans la même boucle, il faut ajouter donc le calcul pour le P de PID

Remarque : La variable kp doit être placé comme ceci devant la boucle pour la définir dès le départ et dont la valeur recommandée est 28.

On va multiplier l'erreur avec le coefficient de proportionnalité kp



Le résultat de cette multiplication va être comparé à 0 : s'il est inférieur à 0, cela signifie que le robot tome en avant et donc les moteurs A et B doivent aller vers l'avant mais s'il est supérieur à 0 alors les moteurs doivent donc aller en arrière.

Le résultat trouvé va aussi donner la valeur de la puissance que les moteurs doivent prendre, qui est donc en fonction de la valeur lumineuse mesurée au début. Pour cette étape, le signe du résultat n'est pas considéré, il n'y a que la valeur de ce résultat qui sera prise en compte.

Il a fallu plusieurs essais pour trouver une valeur de kp qui convienne le mieux. Mais généralement quand la valeur de kp est inférieure à 28, la puissance n'est pas assez forte pour rattraper le robot, et quand elle est supérieure à 28 la puissance est trop forte et fait tomber le robot. 28 semble être le juste milieu.

Pour arriver à un programme comme celui-ci, il est primordial de bien comprendre le principe du contrôle proportionnel.

Observation :

Avec un programme qui ne prend en compte que le controle proportionnel, le résultat, comme vous allez pourvoir le constatez, est un succès. Par rapport à tous les programmes effectués auparavant ici le robot arrive enfin à rester en équilibre.

Mais on voit aussi que le robot est parfois un petit peu brusque et peut tomber et donc nécessite plus de précision.

Voir clé USB pour les vidéos tests.

Test 2 :

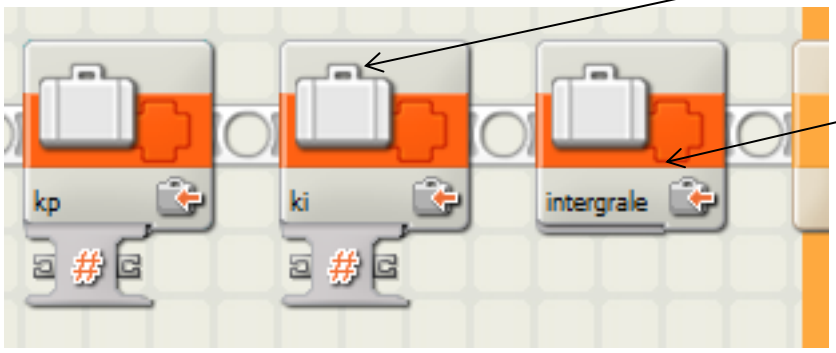
Ce qui nous amène au deuxième test, qui va ajouter à la partie P du PID Controller, le I de Integral.

Principe :

L'erreur entre la consigne et la mesure est ici intégrée par rapport au temps et multipliée par une constante qu'il faudra aussi déterminer en fonction du système.

Lors d'un simple contrôle proportionnel, il subsiste une erreur statique. Lorsque le système s'approche de sa consigne, l'erreur n'est plus assez grande pour faire avancer le moteur. Le terme intégral permet ainsi de compenser l'erreur statique et fournit, par conséquent, un système plus stable en régime permanent. Plus Ki est élevé, plus l'erreur statique est corrigée.

Programme avec explication :

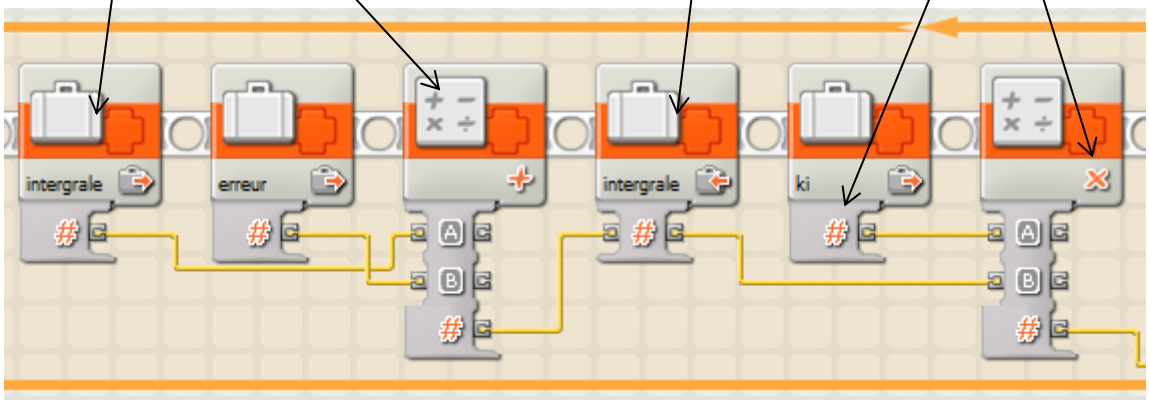


Il faut d'abord rajouter en début de programme et à la suite de kp, la variable ki dont la valeur recommandée est 0.5 et la variable intégrale dont la valeur de départ est 0

Après, à la suite, il faut prendre cette intégrale qui est nulle et lui ajouter la valeur de l'erreur

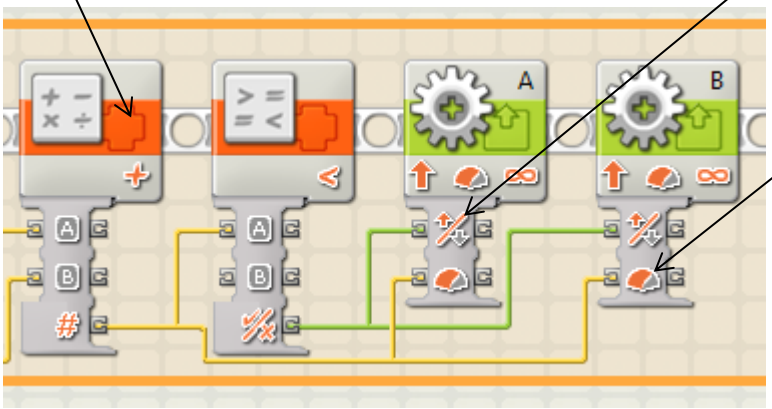
Le résultat sera la nouvelle valeur que prendra l'intégrale.

Et cette nouvelle intégrale, on la multiplie par la constante ki



Le résultat de cette multiplication est ensuite ajouté au résultat précèdent de la multiplication de l'erreur avec kp

Et après c'est la même chose que précédemment, le résultat est comparé à 0 pour donner la direction des moteurs et il leurs donne aussi la valeur de leur puissance.



Rappel : cette partie du programme n'est pas répété deux fois dans le programme, elle n'apparait qu'une fois à la fin de chaque programme.

Vous pouvez effectuer vous-même si vous le souhaitez plusieurs essais pour trouver une valeur de kp qui vous convient le mieux. 0,5 semble être le juste milieu après plusieurs tests dans ce cas aussi.

Observation :

Le robot tient parfaitement en équilibre et ne tombe pas à moins qu'on le bouscule bien sûr. L'intégrale complète la partie proportionnelle pour un résultat presque parfait.

Voir clé USB pour les vidéos tests.

Test 3 :

Maintenant pour compléter le contrôle par PID il faut ajouter à tout cela la partie D de PID, soit la dérivée.

Principe :

Celui-ci consiste à dériver l'erreur entre la consigne et la mesure par rapport au temps et à le multiplier lui aussi par une constante.

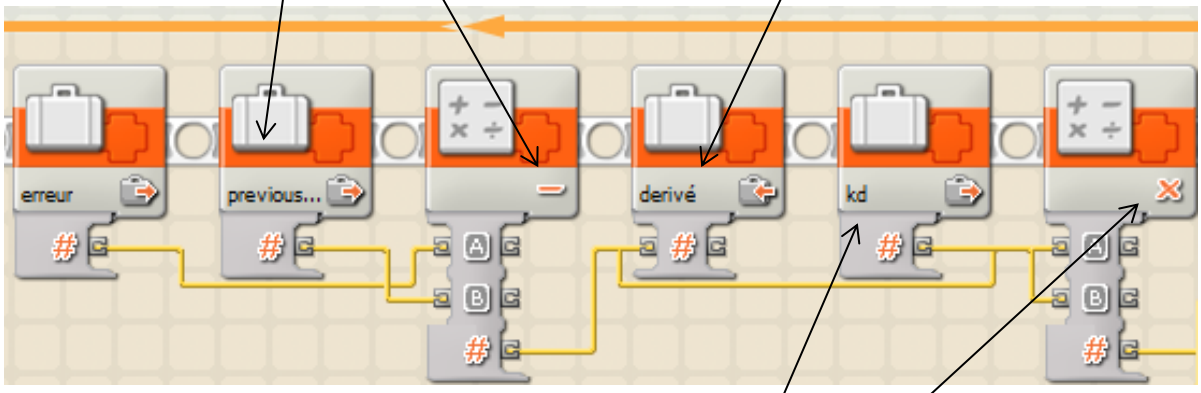
Le contrôle PI peut amener à un dépassement de la consigne, ce qui n'est pas toujours très souhaitable (exemple du robot qui tombe en arrière, il est rattrapé et rétablit non pas à la verticale mais un peu plus en avant). Le terme dérivé permet de limiter cela. Lorsque le système s'approche de la consigne, ce terme freine le système en appliquant une action dans le sens opposé et permet ainsi une stabilisation plus rapide.

Programme avec explication :

Il faut d'abord rajouter encore une fois en début de programme et à la suite de kp, et ki, la variable kd dont la valeur recommandée est 10 et aussi la variable previous-erreur dont la valeur de départ est 0

Après il faut soustraire à l'erreur la previous-error qui est en fait l'erreur d'avant, et qui est de valeur 0 au début car il n'y a pas encore eu d' « avant ».

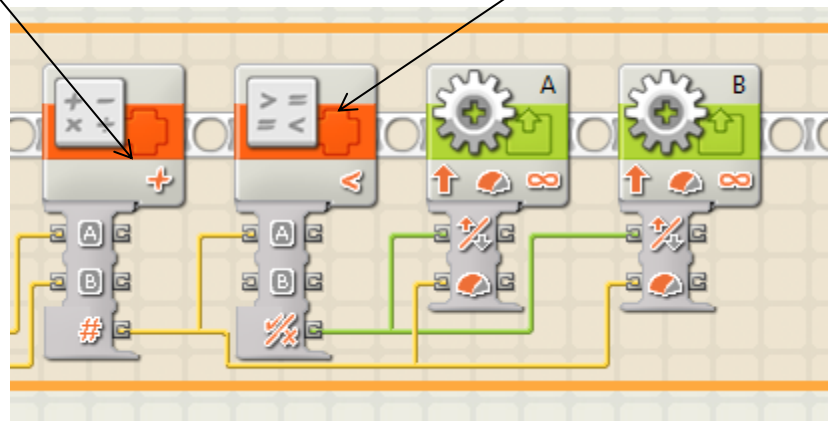
Le résultat de la soustraction devient la dérivée.



Et cette dérivée on la multiplie par le coefficient kd

Le résultat de cette multiplication est ensuite ajouté au résultat précèdent : de l'addition de la multiplication l'erreur par kp avec la multiplication de l'intégrale par ki. Vous pouvez constater qu'en fait les résultats de toutes les étapes, p, i, d, sont additionnés

Et après c'est encore la même chose que tout à l'heure, ce résultat est comparé à 0 pour donner la direction des moteurs et il leurs donne aussi la valeur de leur puissance.



Rappel : cette partie du programme n'est pas répété trois fois dans le programme, elle n'apparait qu'une fois à la fin de chaque programme.

Vous pouvez effectuer vous-même si vous le souhaitez plusieurs essais pour trouver une valeur de kd qui vous convient le mieux. 10 semble être la meilleure valeur.

Observation : Avec la succession des étapes P, I et D, le robot arrive maintenant à être auto balancé comme un segway ! Il parvient finalement à rester en équilibre !

La difficulté ici est qu'il faut absolument bien comprendre le principe du contrôle par PID et de l'asservissement sinon il est difficile de corriger le programme s'il y a des problèmes.

Voir clef USB pour les vidéos tests.

Pour APPROFONDIR

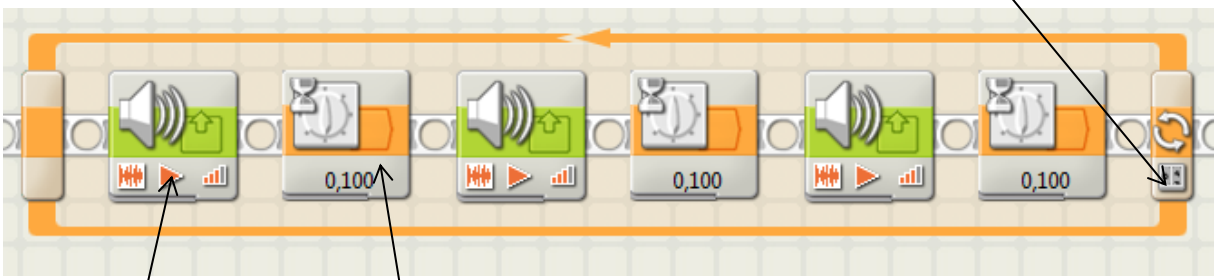
Le résultat est bon avec ce PID programme mais pour un seuil de luminosité à respecter donné dans le programme lui-même, (c'était la constante « eq » déterminant la valeur du seuil de luminosité)

Mais alors comment faire si vous voulez mettre en marche le robot quelque part où vous n'avez pas d'ordinateur pour programmer ?

Il faudrait que le seuil de luminosité à respecter soit déterminé par le robot lui-même. C'est-à-dire qu'en début de programme le robot va devoir déterminer lui-même le seuil à respecter pendant que vous allez le maintenir à la verticale parfaitement.

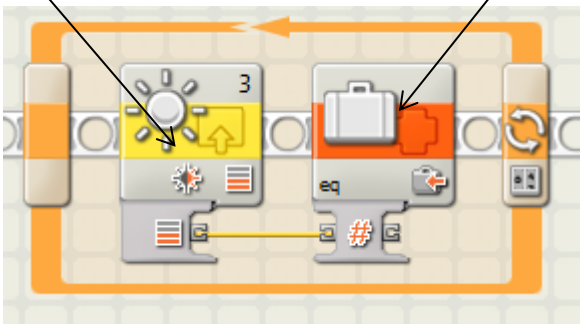
Programme avec explication:

En tout début de programme il faut ajouter cette boucle, qui n'est pas infinie cette fois, mais qui a un compteur de 3, c'est-à-dire qui va répéter trois fois l'action à l'intérieur

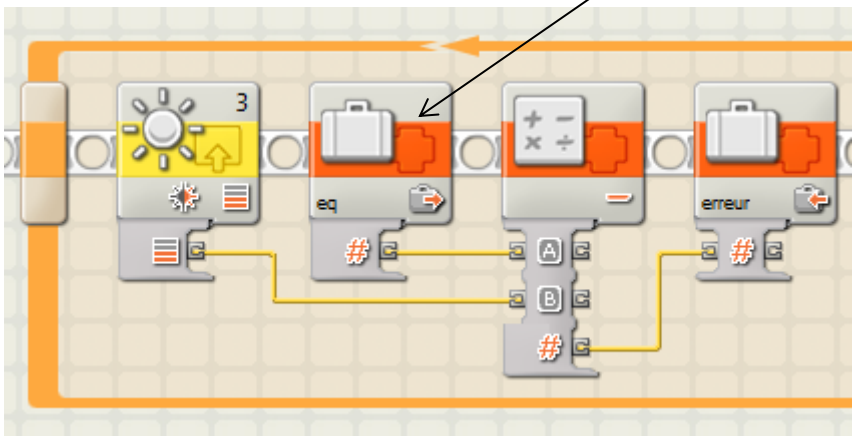


Dans cette boucle on va désigner un son qui va être répété trois fois avec 0.1 d'écart entre chaque son

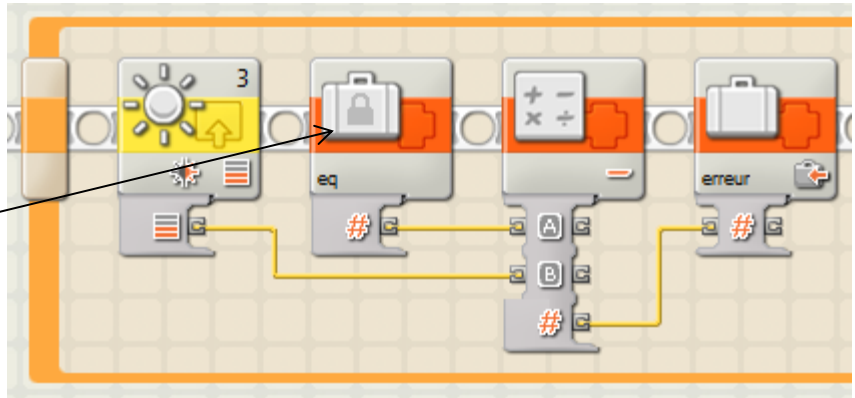
En même temps une autre boucle avec un compteur de 3 va assigner à la valeur que perçoit le capteur photosensible, quand vous maintenez le robot à la verticale, à la valeur du seuil à respecter « eq »



La différence établie maintenant entre le seuil de luminosité à respecter et celle mesurée pour assigner l'erreur (comme il est dit dans le premier test avec la partie P de PID) se fait maintenant avec une variable « eq » …



… et non plus avec une constante comme à l'origine du programme



Il faut bien faire attention aux changements que l'on applique sinon ils peuvent détériorer le programme et le rendre défectueux. *Voir clef USB pour les vidéos tests.*

Et voilà, si vous avez bien suivi toutes ces étapes vous avez maintenant un RobotSegwayNXT fonctionnel !

Conclusion :

Le robot devrait, après ces étapes et après plusieurs essais, fonctionner correctement. Le capteur lumineux et le meilleur capteur fourni dans la boite NXT pour créer ton propre Segway. Or, un capteur plus avancé aurait peut-être permis un fonctionnement encore plus précis du robot.

Un capteur gyroscopique, par exemple, serait une possibilité. Il donne des résultats plus précis car il se base sur son propre équilibre et ne dépend pas de la luminosité de ta pièce, et ce capteur envoie les résultats à ta brique plus vite donc les possibilités de chute du robot sont réduites. Mais là, c'est à toi de jouer pour essayer de programmer ton robot avec un capteur gyroscopique !

Capteur gyroscopique

Sources texte :

http://www.segway.ch/fr/infos/funktionsweise.php

https://fr.wikipedia.org/wiki/Segway_PT

Source principale du principe d'asservissement par PID :
http://lyceejdarc.org/autodoc/cours/003%20T%20STI2D/Technologie%20transversale/Asservissement/020%20Ressources%20documentairs/pid.pdf

Sources Image :

http://www.segway.ch/fr/infos/funktionsweise.php

https://www.msu.edu/~luckie/segway/i180/i180_3shot.jpg

http://docplayer.fr/docs-images/17/118711/images/13-0.png

http://wheel-rider.com/img/cms/Robstep%20M1%20mini%20gyropode%20%C3%A9lectrique.png

http://www.segway.fr/file/content/c05861212e5701e0781d8a86c52c2d8f.jpeg

Les Images du programme sont toutes des captures d'écran des programmes que nous avons créés nous-même et celles du robot sont aussi du robot que nous avons personnellement construit et modifié au cours du temps, photos prises en classe, au laboratoire ou à nos domiciles.

Axel Lovera et Alistair Laurière. 1ère S5