**Projet – Intelligence artificielle au service de la mobilité**

*Base robotique mobile*

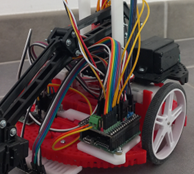
***Objectif : Conception d’une base robotique mobile dotée d’intelligence artificielle***

Ce projet vise à mettre en place une **base robotique mobile** capable de **parcourir un circuit** dans sa totalité **de façon autonome**.



Le parcours sera balisé à l’aide de **parois verticales d’une hauteur d’environ 50 cm** :

On dispose d’une **base robotique**, reposant sur le châssis POLOLU Romi, sur laquelle on a implanté une **carte Pyboard** programmable en Python et une **carte de commande moteurs**. <https://www.pololu.com/category/202/romi-chassis-and-accessories>



**Remarque : L’usage d’une base robotique équivalente est tout à fait envisageable.**

Cette base robotique mobile devra être **dotée d’intelligence artificielle**, autrement dit :

* le **comportement de cette base robotique** devra être décrit à l’aide de **perceptrons** (de neurones) ;
* **cette base robotique devra exploiter une base d’apprentissage** (un jeu de données) **afin d’établir le comportement qu’elle devra adopter** dans les différentes situations rencontrées (à l’issue d’un apprentissage supervisé).

***Organisation :***

Au cours du projet, il y aura une alternance entre des **phases actives de projet** (démarche de projet) et des **phases d’activité** (présence d’un énoncé à suivre).

Durant les phases d’activité, il faudra concevoir **un compte-rendu numérique par personne**. Dans les phases de projet, les **documents communs** seront à rendre par le groupe.

Le travail conduisant progressivement à **répondre à la problématique** sera le suivant :

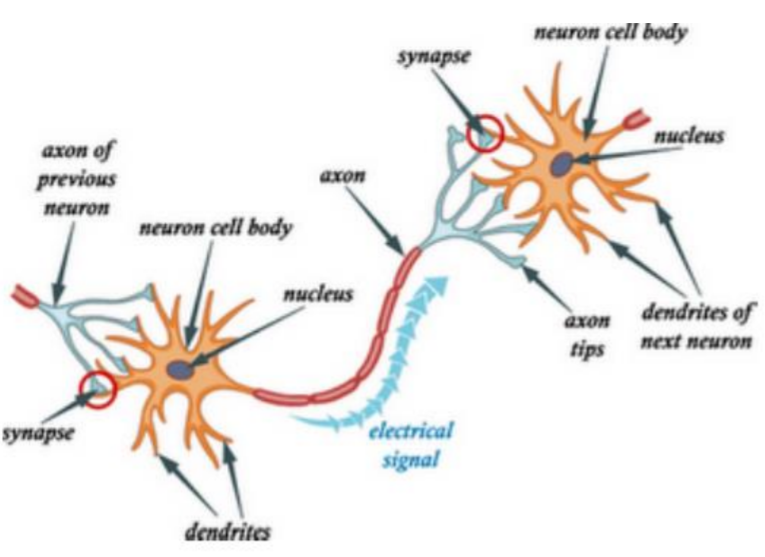
1. Aborder la **notion** d’intelligence artificielle (notion de **perceptron** et d’**apprentissage supervisé**) ;
2. Mettre en place une solution d’**évitement de collision frontale** lors d’un mouvement de translation ;
3. Mettre en place une solution de **positionnement angulaire dans une direction ne présentant pas d’obstacle** ;
4. Mettre en place la solution de **déplacement afin d’effectuer la totalité du parcours**.

Chaque étape comportera une phase d’**analyse**, de **modélisation**, d’**expérimentation** puis d’analyse des **écarts**.

**Remarque : Ce projet est à destination d’élèves de terminale spécialité SI.**

**Commentaire : Ce document a pour objectif de guider les élèves au cours du projet. Il ne faut cependant pas nécessairement s’imposer de le suivre à la lettre, ni de chercher à réaliser l’ensemble des travaux proposés**

***I. Biologie vs intelligence artificielle***



En biologie, un **neurone** est une cellule du système nerveux spécialisée dans la communication et le **traitement de l’information**.

Un neurone comporte de **nombreuses ramifications d’entrée** (dendrites) d’où proviennent les **informations** (issues des synapses).

Il génère sur sa **sortie** (axone) une information à destination des neurones suivants.

Les synapses en contact les dendrites d’un neurone n’ont **pas toutes le même « poids »**, autrement dit **les entrées n’ont pas la même influence sur la sortie** du neurone.

Conceptuellement l’**intelligence artificielle** (l’IA) consiste à **mimer ce phénomène biologique**.

**Limitation : Afin de ne pas alourdir le propos, la notion de fonction d’activation d’un neurone ne sera volontairement pas décrite ici.**

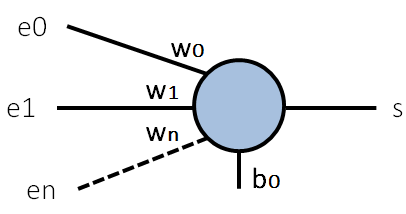
L’IA décrit le neurone (autrement nommé **perceptron**) sous la forme d’une relation d’entrées / sortie du type :

**s = w0 × e0 + w1 × e1 + … + wn × en + b0**

**w0**, **w1**, …, **wn** sont nommées les **poids**

**b0** est nommé le **biais**

Il est commode de représenter le neurone sous **forme graphique** (à gauche les entrées et à droite la sortie) :



Chacune des entrées du neurone est **pondérée**. Ces entrées n’ont donc **pas la même influence** sur la sortie du neurone.

A l’image de son **jumeau biologique**, l’influence des entrées **e0**, **e1**, …, **en** **dépend** respectivement de la valeur des poids **w0**, **w1**, …, **wn** (plus un poids à une valeur élevée, plus l’entrée associée influencera la sortie du perceptron).

**Remarque : le projet ne nécessite pas la mise en place d’un réseau de neurones à couches cachées car seul le problème de la régression sera utile au projet.**

**UNIQUEMENT pour les curieux : on considère ici que la fonction d’activation f du neurone est de nature affine (on a ici s = f(w0 × e0 + w1 × e1 + … + b0) = w0 × e0 + w1 × e1 + … + b0). Cette fonction d’activation affine sera suffisante dans ce projet car on ne cherchera pas à faire de la classification mais "uniquement" de la régression linéaire.**

**UNIQUEMENT pour les curieux : Le choix d’une fonction d’activation affine ne permet d’ailleurs pas de mettre en place des réseaux de neurones à couches cachées (qu’une classification évoluée réclamerait). En effet, la mise en cascade de neurones dotés de fonctions d’activation affines se décrit à l’aide d’un seul neurone à fonction d’activation affine.**

***II. Mise en place d’un perceptron : évitement de collision frontale***

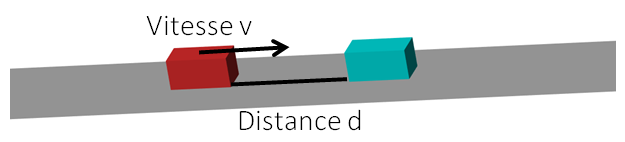
***a. Modélisation***

On souhaite **mettre en place un perceptron** au sein du système de traitement numérique de la base robotique mobile afin de **décrire son comportement à l’approche d’une paroi verticale**.

La base robotique mobile est ici capable de **mouvements de translation** dans un environnement comportant des **parois verticales**.

**Base robotique mobile**

**Paroi verticale**



Le robot se doit, entre autres, de **se déplacer en translation en direction d’une paroi verticale** mais **sans rentrer en collision** avec cette dernière.

Pour cela, le robot dispose d’un **capteur de distance** afin de mesurer la distance **d** le séparant de la paroi.

On souhaite faire en sorte que le robot **adapte sa vitesse** **en fonction de la distance** **d** le séparant de la paroi **selon une relation mathématique de type fonction affine**.

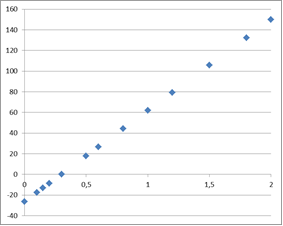
**Remarque :** dans la suite de l’énoncé, on considère que **v** est le nom d’une **variable numérique** dont la valeur, comprise entre -255 et 255, est représentative de la **vitesse** et du **sens de déplacement** du de la base.

**v = a × d + b**

On choisit ici de noter **a** : **w3** et **b** : **b3** la relation devient donc : **v = w3 × d + b3**

***Q1.*** **Expliquer** ce qui permet d’envisager l’usage d’un perceptron pour répondre à ce besoin d’évitement de collision. **Dessiner**, sous forme graphique, le perceptron associé à la relation mathématique liant **v** et **d**. **Donner** le nombre d’entrées de ce perceptron et l’intitulé des poids associés. **Indiquer** si ce perceptron dispose d’un biais, si oui, **indiquer** l’intitulé de ce biais.

Dans un premier temps, pour simplifier le raisonnement, **on s’impose arbitrairement que** la valeur numérique **v** et la distance **d** soient régies par les couples de valeurs **(d , v)** suivants :



|  |  |
| --- | --- |
| **Distance d en m** | **Donnée numérique v représentative de la vitesse** |
| 0 | -26.471 |
| 0,1 | -17,6471 |
| 0,15 | -13,2353 |
| 0,2 | -8,82353 |
| 0,3 | 0 |
| 0,5 | 17,64706 |
| 0,6 | 26,47059 |
| 0,8 | 44,11765 |
| 1 | 61,76471 |
| 1,2 | 79,41176 |
| 1,5 | 105,8824 |
| 1,8 | 132,3529 |
| 2 | 150 |

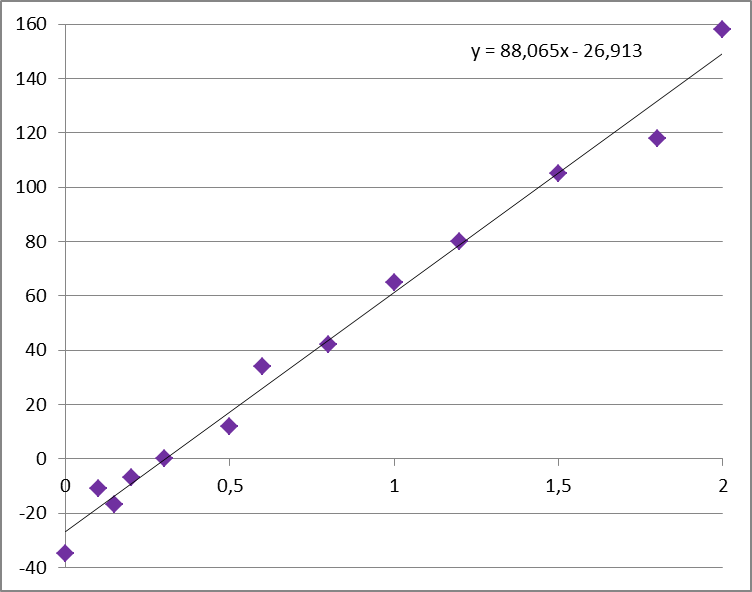
***Q2.*** A l’aide d’un tableur, **tracer** **v** en fonction de **d** (**faire** une capture d’écran). **Déterminer** l’équation (du type y = a x + b) liant **v** en fonction de **d**. **Indiquer** la valeur du poids **w3** et du biais **b3** du perceptron associé.

***Q3.*** **Déterminer** la valeur de **v** pour les valeurs de **d** suivantes :

1.9m 0.9m 0.3m 0.05m

**Indiquer** le comportement du robot dans chacun de ces cas.

**Remarque : les points associés aux couples de valeurs (d , v) peuvent ne pas être alignés, il serait, par exemple, tout à fait envisageable de considérer les couples de valeurs suivantes et puis de chercher la courbe de tendance (fonction affine) qui décrit le plus correctement possible le comportement attendu :**

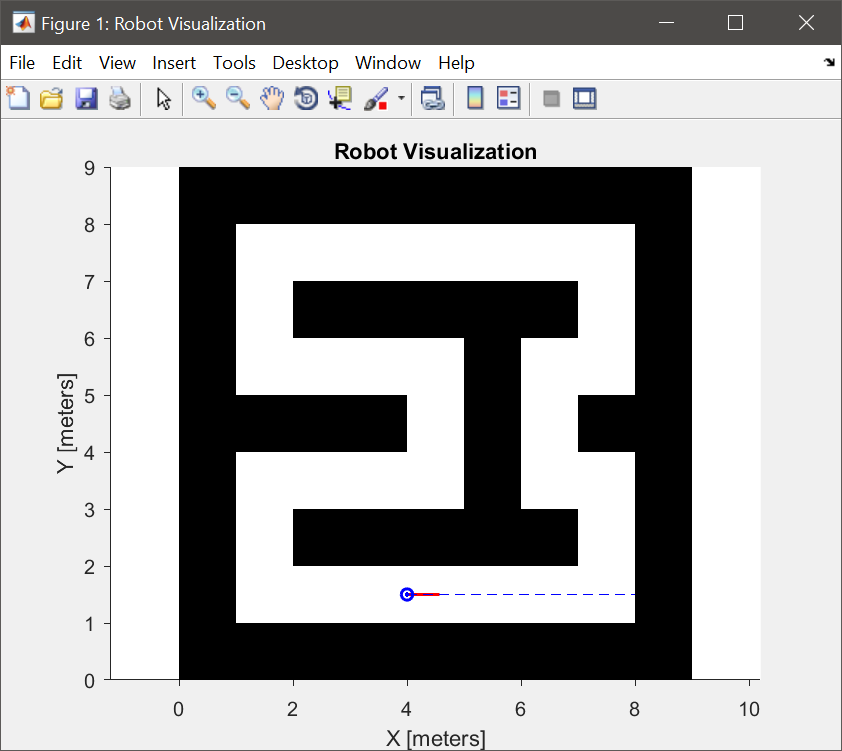


|  |  |
| --- | --- |
| **Distance d en m** | **Donnée numérique v représentative de la vitesse** |
| 0 | -35 |
| 0,1 | -11 |
| 0,15 | -17 |
| 0,2 | -7 |
| 0,3 | 0 |
| 0,5 | 12 |
| 0,6 | 34 |
| 0,8 | 42 |
| 1 | 65 |
| 1,2 | 80 |
| 1,5 | 105 |
| 1,8 | 118 |
| 2 | 158 |

**La recherche de l’équation de la courbe de tendance (ici une fonction affine) sera à terme réalisée par la base robotique elle-même (voir la partie portant sur l’apprentissage supervisé).**

On souhaite à présent **modéliser le comportement du robot** lorsque ce dernier va s’approcher de la paroi. Pour cela on va utiliser Matlab / Simulink.

Dans cette simulation Matlab /Simulink, le robot est représenté vu de dessus sous la forme d’un **cercle bleu** :



Dans la simulation :

* le **trait rouge** est représentatif de la direction dans laquelle se déplace le robot ;
* le **trait en pointillés bleu** est représentatif de la **distance mesurée** par le capteur présent sur le robot.
* les **zones blanches** ne comportent **pas d’obstacles** ;
* les **zones noires** sont des **parois verticales** ;
* lors du déplacement, la **trajectoire du robot** apparait sous la forme d’un **ensemble de points bleus** :



Voici les étapes à suivre pour **modéliser le déplacement du robot en présence de parois verticales** :

***Q4.*** **Ouvrir** Matlab / Simulink, **positionner**, **interconnecter** puis **paramétrer** les blocs décrits ci-après. **Faire** une capture d’écran du modèle. **Comprendre** le rôle des blocs. **Nommer** le fichier **evitecollisionfrontale.slx**.

Remarque : pour positionner un bloc il suffit de double-cliquer sur le fond blanc de la fenêtre Simulink puis de saisir le nom de bloc à positionner.

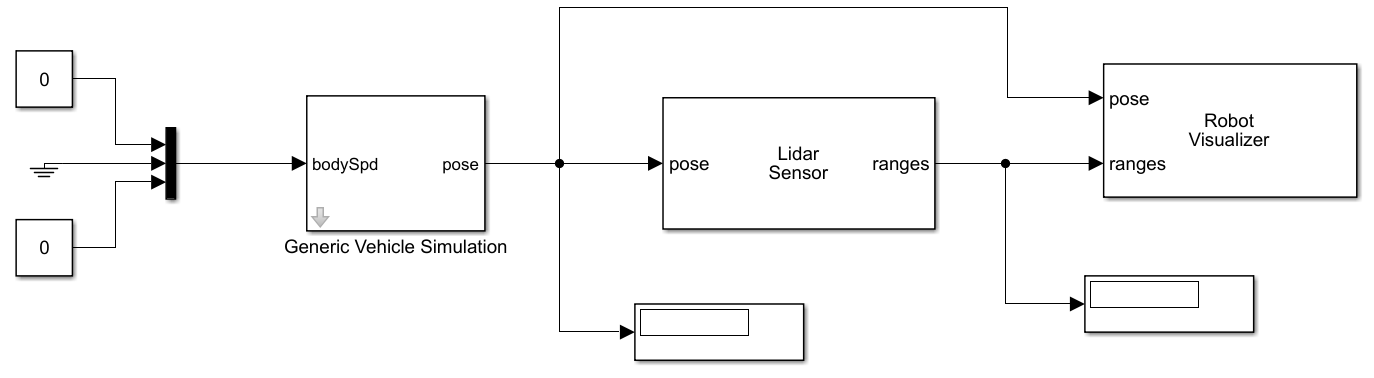
**Vitesse de déplacement   
du robot en m/s**

**Capteur  
de distance présent sur le robot**

**Affichage de la position du robot (x, y et position angulaire)**

**Robot**

**Bloc de visualisation de la trajectoire du robot**



**Affichage de la distance mesurée par le capteur**

|  |  |
| --- | --- |
| Détail des blocs : | Paramétrage des blocs : |
| **Constant** |  |
| **Ground**  Ce bloc est représentatif d’une entrée mise à 0. | Pas de paramétrage à mettre en place. |
| **Mux**  Ce bloc permet de regrouper des signaux sur une même liaison. |  |
| **Generic Vehicle Simulation**  Ici la pose initiale du robot mobile est X égal 4m et Y égal 1.5m et sa position angulaire est de 0 radian.  Le robot (rond bleu) est alors orienté vers la droite comme l’indique le trait rouge : |  |
| **Display** | Pas de paramétrage à mettre en place si ce n’est un redimensionnement pour voir les valeurs. |
| **Lidar Sensor**  **maplabypetit** : correspond à **l’environnement comportant des obstacles** (en noir) au format BinaryOccupancyGrid :  **[0, 0] (valeur par défaut)** : correspond à la position du / des capteurs sur le robot (par défaut en position centrale sur le robot).  **[0]** : correspond au positionnement angulaire (en radian) du capteur sur le robot (ici le capteur est orienté dans l’axe du robot comme l’indique le trait en pointillé bleu, 0 radian) :  **8** : correspond à l’étendue de mesure du capteur de distance (ici 8 mètres). | Important : Pour générer le fichier **maplabypetit** au format BinaryOccupancyGrid à partir de l’image **maplabypetit.png**, il faut exécuter le script **creationmap.m** comportant la ligne :  maplabypetit = robotics.BinaryOccupancyGrid(rgb2gray(imread('maplabypetit.png')) < 0.5);  Veiller à se placer dans le répertoire où se situe le fichier **maplabypetit.png**.  Il est possible de modifier le fichier **maplabypetit.png** (sur Paint par exemple, sachant que la longueur d’un pixel est équivalent à 1m) afin d’ajouter des parois, en ajoutant des pixels noirs, puis de relancer le script **creationmap.m**.  **Remarque : Si vous travaillez avec une version postérieure à Matlab 2018b la commande est : maplabypetit = binaryOccupancyMap(rgb2gray(imread('maplabypetit.png')) < 0.5);** |
| **Robot Visualizer**  **0.1** : correspond à la dimension du robot dans la visualisation.  **maplabypetit** : correspond à **l’environnement comportant des parois verticales** (en noir) au format BinaryOccupancyGrid. Il est **obligatoire** que ce paramètre soit le même que celui du bloc **Lidar Sensor**.  **Lidar sensor offset** et **Scan angles** doivent **obligatoirement** avoir les mêmes valeurs que ceux du bloc **Lidar Sensor**. |  |

***Q5.*** **Lancer** la simulation. **Relever** les valeurs de la pose du robot (**x**, **y**, **position angulaire**). **Indiquer** le comportement du robot. **Faire** des captures d’écran issues du modèle.

***Q6.*** **Modifier** la vitesse de translation à **0.1** m/s. **Relancer** la simulation (sur 10 secondes) et **indiquer** la pertinence des résultats obtenus dans les blocs Display. **Indiquer** la distance **d** entre le robot et l’obstacle à la fin de la simulation. **Nommer** les liaisons du modèle Simulink (pour cela il faut double-cliquer sur la liaison à nommer). **Faire** des captures d’écran issues du modèle.

***Q7.*** **Mettre en place**, dans le modèle Simulink, le perceptron décrit dans les questions précédentes permettant le calcul de la vitesse du robot en fonction de la distance **d** le séparant de la paroi verticale. Pour cela **utiliser** les blocs ci-dessous. **Nommer** le fichier **evitecollisionfrontale\_perceptron.slx**.

- Constant :



- Product :



- Sum :



Remarque : la relation entre la valeur numérique **v** représentative de la vitesse et la vitesse réelle du robot sera considérée comme étant la suivante :

**vitesse = v / 255**

Pour cela utiliser le bloc :

- Gain :

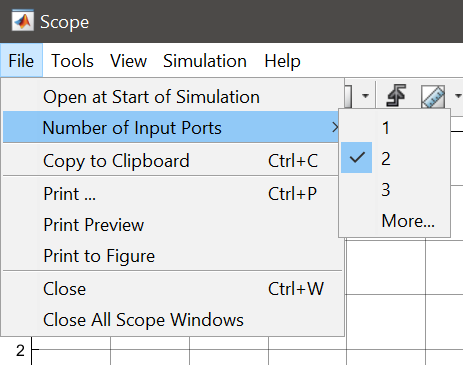


***Q8.*** **Saisir** un temps de simulation infini (**inf**) puis **lancer** la simulation. **Décrire** précisément le comportement du robot. **Relever** les valeurs de la pose du robot (**x**, **y**, **position angulaire**) ainsi que la distance le séparant de la paroi verticale à l’issue du mouvement de translation du robot. **Démontrer** par calcul cette valeur. **Faire** des captures d’écran issues du modèle.



***Q9.*** **Visualiser** l’évolution sur 20 secondes de la distance **d**, de la position et de la vitesse du robot dans le temps en ajoutant un scope.

Remarque : voici la démarche pour ajouter des entrées à un Scope sur Simulink :



***Q10.*** **Modifier** arbitrairement le poids **w3** et le biais **b3** du perceptron et **commenter** le comportement du robot en analysant les relevés temporels de la distance **d**, de la position et de la vitesse du robot dans le temps :

**w3** = 20 et **b3** = 0

**w3** = -20 et **b3** = 0

***b. PROJET : Implémentation en Python du perceptron sur une base robotique mobile réelle***

Dans cette partie, la répartition du travail au sein du groupe d’élèves est libre.

**On souhaite à présent implémenter le perceptron, modélisé précédemment, en Python sur la base robotique réelle.**

Pour cela trois étapes vont être nécessaires.

**Etape 1 : Mesure de la distance séparant le robot de la paroi verticale**

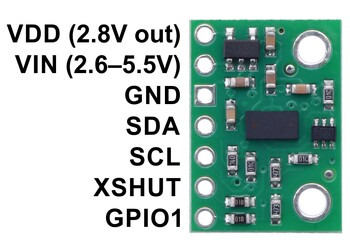
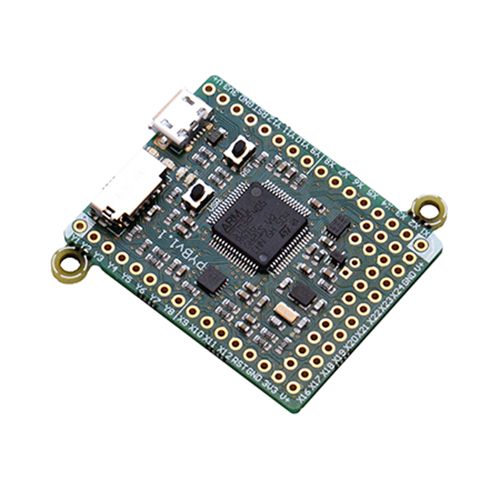
Cahier des charges : On s’impose que le capteur de distance ait une étendue de mesure de 5cm à 2m minimum avec une résolution de minimum 1mm. Ce capteur sera à implanter sur la base robotique. Il devra donc être compatible avec la carte Pyboard.

***Q11.*** Après l’analyse des informations issues du lien ci-dessous, **justifier** le bien fondé du choix du capteur VL53L0x. **Rédiger** la réponse.

<https://www.gotronic.fr/art-capteur-de-distance-2m-2490-25155.htm>

***Q12.*** A l’aide des vues ci-dessous, **dessiner** le schéma de câblage du capteur avec la Pyboard.

*Description des broches de connexion du capteur VL53L0x avec la Pyboard :*



**VIN** : 3.3V

**GND** : GND

**SDA** : SDA

**SCL** : SCL

**XSHUT** : Y10

***Q13.*** A l’aide de l’extrait de code Python ci-dessous, **mettre en place** la solution mesure de distance.

**Procéder** à des essais de mesure. **Enregistrer** le programme sous le nom **mesure\_distance\_1\_capteur.py**.

Remarques : les fichiers **init\_vl53l0x.py** et **vl53l0x.py** doivent être placés sur la pyBoard.

**Programme mesure\_distance\_1\_capteur.py :**

**from pyb import Pin  
from machine import I2C, Pin  
import init\_vl53l0x  
import time  
  
i2c = I2C(scl=Pin('X9'), sda=Pin('X10'))  
  
capteur\_centre\_xshut = Pin('Y10', Pin.OUT)  
  
tofl\_centre = init\_vl53l0x.init\_capteurs\_centre(i2c, capteur\_centre\_xshut)  
  
tofl\_centre.start()  
  
while True:**

**distance\_centre = tofl\_centre.read()/1000  
   
 print("Centre : " + str(distance\_centre))  
  
 time.sleep\_ms(500)**

***Q14.*** **Etablir** puis **mettre en place** un protocole expérimental simple permettant de **vérifier** la pertinence des mesures de distances réalisées avec le capteur VL53L0x.

***Q15.*** **Proposer** et **mettre en place** une solution visant à implanter le capteur de distance sur la base robotique.

**Etape 2 : Mouvement de translation de la base robotique**

Cahier des charges : On se doit de connaitre la relation entre la valeur numérique **v**, représentative de la vitesse, et la vitesse réelle du robot.

***Q16.*** **Mettre en place** un programme Python sur la base robotique afin que le robot mobile effectue un mouvement de translation à vitesse constante. **Enregistrer** le programme sous le nom **cde\_moteurs.py**.

***Q17.*** **Etablir** et **mettre en place** un protocole expérimental simple visant à **établir** la relation entre la valeur numérique **v**, représentative de la vitesse du robot, et la vitesse de déplacement réel du robot. **Rédiger** la réponse et les résultats de l’expérimentation.

**Etape 3 : Implémentation du perceptron d’évitement de collision en Python**

Cahier des charges : Le cahier des charges impose que le perceptron soit implanté en Python.

***Q18.*** **Analyser** puis **compléter** le code ci-après afin de **procéder** à la fusion des codes Python de mesure de distance **mesure\_distance\_1\_capteur.py** et de mise en mouvement en translation de la base robotique **cde\_moteurs.py** dans le but de **mettre en place** le perceptron en Python sur la base robotique.

**Enregistrer** le programme sous le nom **perceptron\_evitecollision.py**.

**Procéder** à des premiers essais.

**Programme perceptron\_evitecollision.py :**

**from pyb import Pin, Timer  
from machine import I2C, Pin  
import init\_vl53l0x  
import time  
  
i2c = I2C(scl=Pin('X9'), sda=Pin('X10'))**

**capteur\_centre\_xshut = Pin('Y10', Pin.OUT)  
tofl\_centre = init\_vl53l0x.init\_capteurs\_centre(i2c, capteur\_centre\_xshut)  
tofl\_centre.start()  
  
...   
  
while True:  
 distance\_centre = tofl\_centre.read()/1000  
   
 w3 = ...  
 b3 = ...  
 v = ...  
   
 if v > 60:  
 v = 60  
 elif v < -60:  
 v = ...  
   
 if v >= 0:  
 Mot\_A\_Sens. ...  
 Mot\_B\_Sens.high()  
 else:  
 v = v \* ...  
 Mot\_A\_Sens. ...  
 Mot\_B\_Sens. ...  
   
 Mot\_A\_Vitesse.pulse\_width\_percent(int(v))  
 Mot\_B\_Vitesse.pulse\_width\_percent(int(v))  
   
 time.sleep\_ms(10)**

***Q19.*** **Etablir** et **mettre en place** un protocole expérimental visant à tracer l’évolution réelle de la position du robot en fonction du temps (l’usage du terminal Thonny est à prévoir) lors de l’approche d’un obstacle.

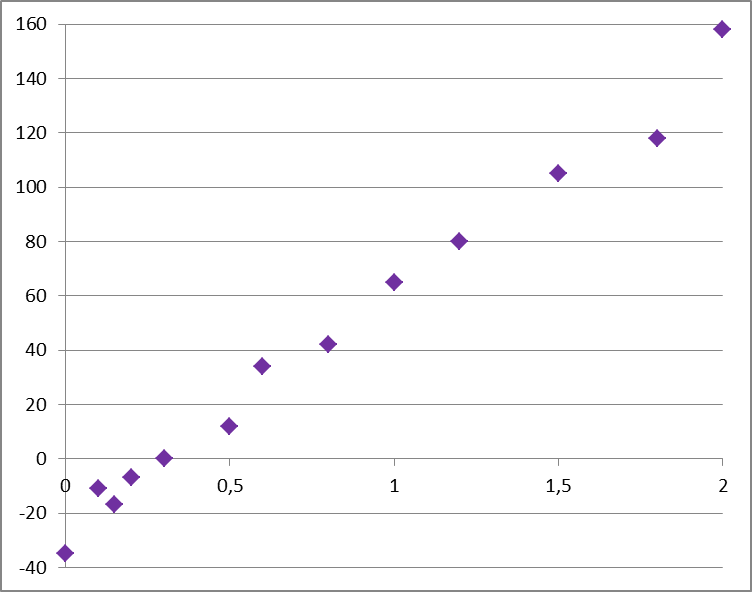
**Rédiger** votre réponse et les résultats de votre expérimentation.

**Etablir** les écarts avec le modèle.

***III. Mise en place de l’apprentissage supervisé***

***a. Modélisation***

|  |  |
| --- | --- |
| Cette partie a pour objectif de mettre en place un **apprentissage supervisé** au sein du modèle Simulink (et à terme au sein du système de traitement numérique de la base robotique mobile).  En effet, on souhaite que les valeurs du poids **w3** et du biais **b3** du perceptron **ne soient plus saisies directement dans le modèle** Simulink mais soient **déterminées à partir de l’ensemble des couples** **(d, v)**.  Durant cette phase d’**apprentissage supervisé**, le poids **w3** et le biais **b3** du perceptron vont être **itérativement modifiés** jusqu’à obtenir le **comportement souhaité**.  Ce comportement est établi par l’ensemble des couples **(d, v)** que l’on nomme la **base d’apprentissage**. |  |



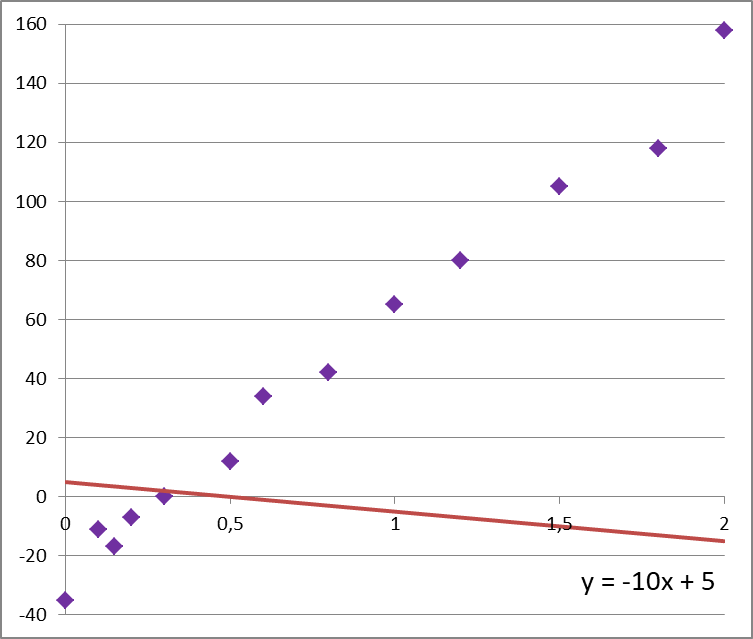
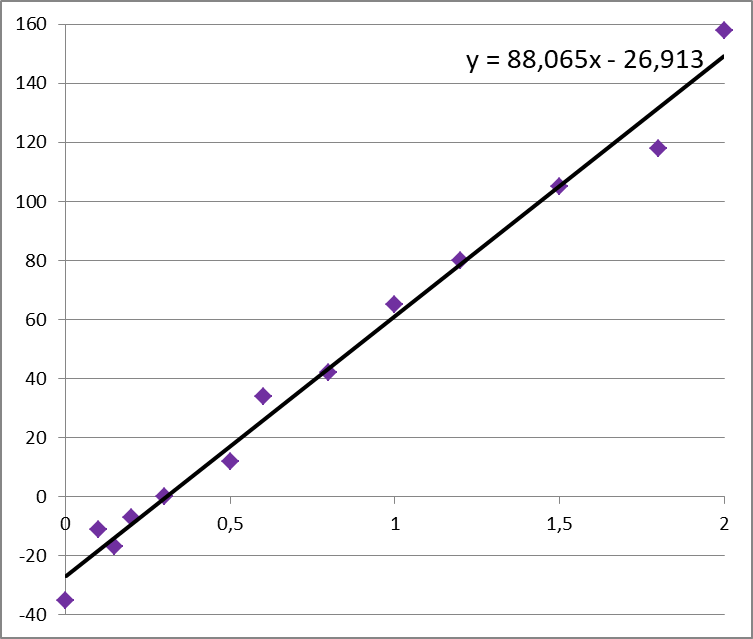
Pour rappel :

* On dispose d’une **base d’apprentissage** (jeu de données **(d, v)**) où **v** est représentative de la vitesse à laquelle le robot doit se déplacer pour la distance **d** le séparant de la paroi.
* On a choisi ici une **description linéaire** dont les poids / biais sont à trouver : **v = w3 \* d + b3**

**Initialement**, le modèle Simulink **ne connait pas la valeur des paramètres w3 et b3**, il lui est donc initialement impossible de tracer la "bonne" droite sur le nuage de point, **à moins de choisir des paramètres au hasard**.

L’objectif est ici de faire en sorte que le modèle Simulink soit capable de **déterminer la valeur des paramètres w3 et b3** à l’issue d’une phase d’apprentissage :

*Paramètres initiaux (choisis arbitrairement) Paramètres finaux (que le modèle doit trouver)*

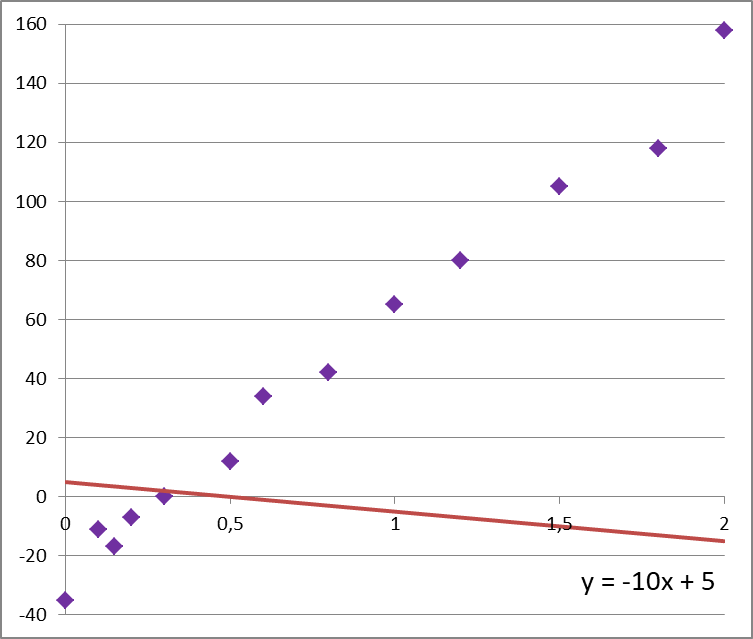
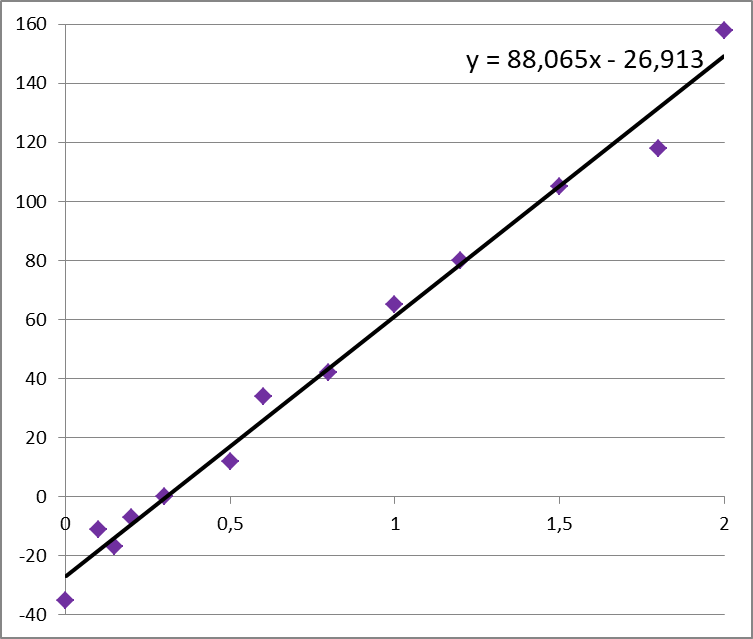


**w3 = -88.06 b3 = 26.91**

**w3 = -10 b3 = 5**

Le modèle Simulink, et à terme la base robotique, va devoir **évaluer** puis **faire évoluer itérativement les paramètres w3 et b3** en cherchant **itérativement à minimiser la longueur des segments** repérés ici en orange.

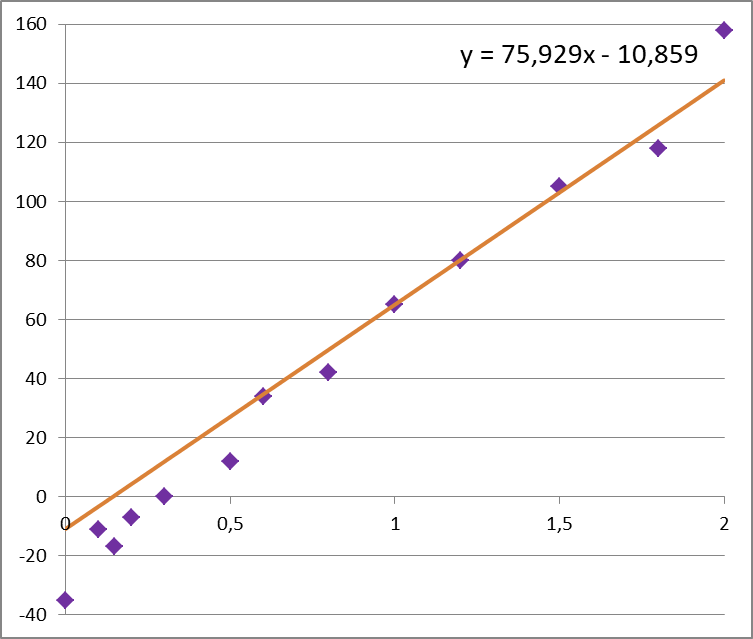
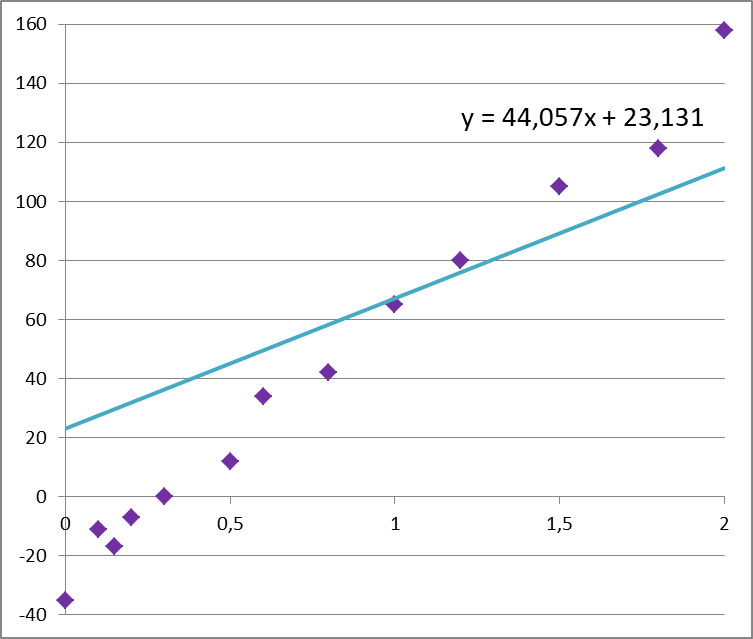
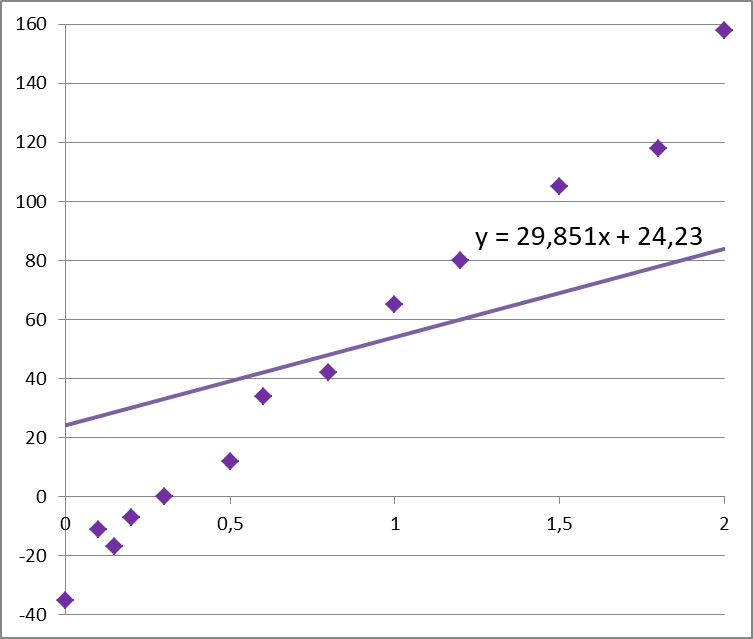
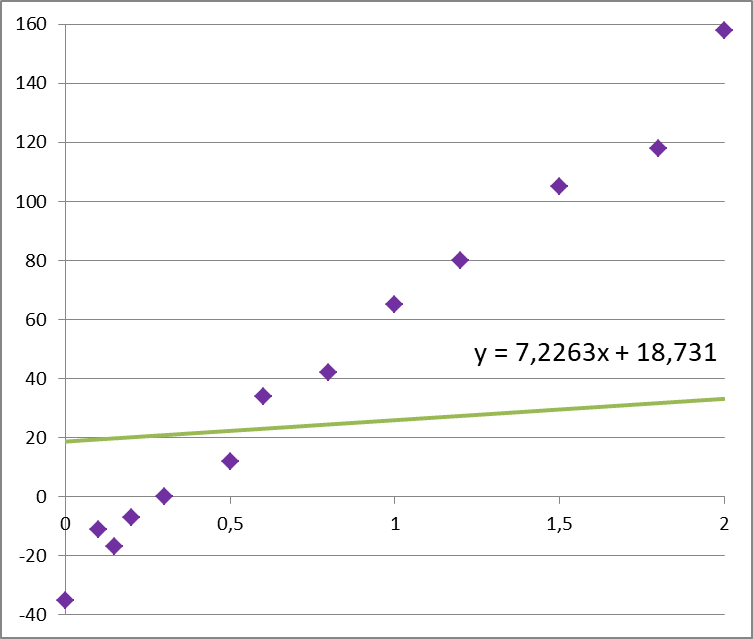
*Paramètres initiaux (choisis arbitrairement) Paramètres finaux (que le modèle doit trouver)*



**w3 = -10 b3 = 5**

**w3 = -88.06 b3 = 26.91**

*Exemples de paramètres intermédiaires obtenus au cours de l’apprentissage supervisé*



L’apprentissage consiste ici en la **recherche itérative d'une droite qui s'ajuste le mieux possible au nuage de points**.

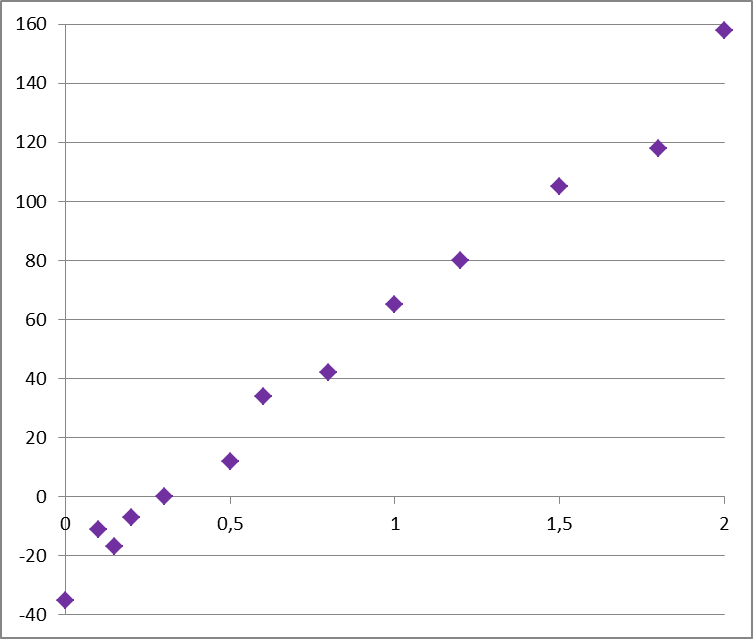
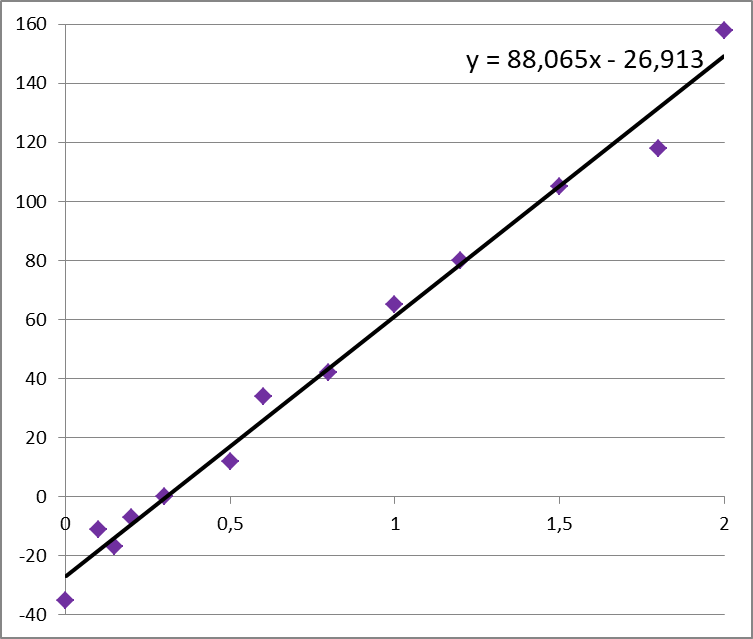
**Limitation : La notion d’« Erreur quadratique » est ici volontairement passée sous silence.**

A l’issue de l’apprentissage supervisé le modèle Simulink (et à terme **la base robotique mobile**) **sera en mesure de prédire le comportement à adopter** :

*AVANT l’apprentissage supervisé APRES l’apprentissage supervisé*

**Possibilité de prédire le comportement à adopter !**

**Impossibilité de prédire le comportement à adopter…**



**???**

**96**

**1,4**

**1,4**

A l'issue de l'apprentissage supervisé, **la base robotique a appris à prédire la vitesse v à laquelle elle devra se déplacer quelle que soit la distance d la séparant de la paroi**.

L’algorithme à mettre en place dans le cadre de **l’apprentissage supervisé** est le suivant :

|  |  |
| --- | --- |
| **0.** On choisit **arbitrairement** la valeur du poids **w3** et du biais **b3** ici :  **w3 = -10,00** et **b3 = 5,00** |  |

|  |  |
| --- | --- |
| **1.** On soumet le perceptron au **premier couple** **(d, v)** de la base d’apprentissage. Puis on calcule la valeur **v** obtenue en sortie du perceptron :  **vobtenu = -10 ,00 × d + 5,00 = 4,00**  On parle ici de **propagation avant**. |  |

|  |  |
| --- | --- |
| **2.** On calcule l’**erreur** entre la sortie **vobtenu** du neurone et la valeur attendue **v** présente dans la base d’apprentissage :  **erreur = -17,64 – 4,00 = -21,64** |  |

|  |  |
| --- | --- |
| **3.** On recalcule une **nouvelle valeur** de **w3** et de **b3** à l’aide des relations suivantes (que l’on admettra dans un premier temps) :  **w3 = w3 + 0,05 × distance × erreur**  **b3 = b3 + 0,05 × erreur**  **w3 = -10,00 + 0,05 × 0,10 × -21,64 = -10,11**  **b3 = 5,00 + 0,05 × -21,64 = 3,92**  On parle ici de **rétro-propagation**. |  |

**Limitations : le programme de SI indique explicitement que les aspects mathématiques ne doivent pas être abordés. On admettra le bien-fondé de ces deux relations mais on tachera de vérifier que le résultat final obtenu est pertinent.**

**Pour les curieux : L’algorithme utilisé ici est basé sur une « descente de gradient stochastique ». Cela signifie que l’on recherche itérativement, pour chacun des couples (d, v), les valeurs de w3 et de b3 permettant de minimiser l’erreur v – vobtenu.**

**Lors d’une « descente de gradient classique », les valeurs de w3 et b3 sont recalculées sur la base de la moyenne des erreurs obtenues à partir de l’ensemble des données présentes dans la base d’apprentissage.**

**Pour les curieux : Je vous conseille vivement, à l’occasion, de survoler le livre DeepMath qui consacre des chapitres entiers à la régression linéaire, à l’algorithme de descente de gradient et à l’apprentissage supervisé :** [**http://exo7.emath.fr/cours/livre-deepmath.pdf**](http://exo7.emath.fr/cours/livre-deepmath.pdf)

Puis on recommence :

|  |  |
| --- | --- |
| **1.** On soumet le perceptron au **premier couple** (distance / vitesse) de la base d’apprentissage. Puis on calcule la valeur **v** obtenue en sortie du perceptron :  **vobtenu = -10 ,11 × d + 3,92 = 2,40**  On parle ici de **propagation avant**. |  |

|  |  |
| --- | --- |
| **2.** On calcule l’**erreur** entre la sortie **vobtenu** du neurone et la valeur attendue **v** présente dans la base d’apprentissage :  **erreur = -13,23 – 2,40 = -15,63** |  |

|  |  |
| --- | --- |
| **3.** On recalcule une **nouvelle valeur** de **w3** et de **b3** à l’aide des relations suivantes :  **w3 = w3 + 0,05 × distance × erreur**  **b3 = b3 + 0,05 × erreur**  **w3 = -10,11 + 0,05 × 0,15 × -15,63 = -10,22**  **b3 = 3,92 + 0,05 × -15,64 = 3,14**  On parle ici de **rétro-propagation**. |  |

***Q20.*** Sachant que tous les couples **(d , v)** présents dans la base d’apprentissage doivent être progressivement parcourus, **décrire**, de la même manière que précédemment, les étapes 1, 2 et 3 suivantes.

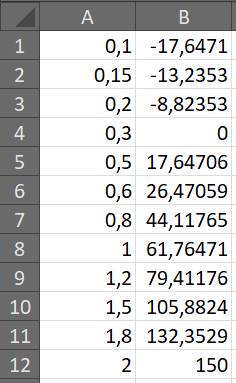
***Q21.*** **Mettre en place** un tableau Excel (selon le modèle ci-dessous) permettant de parcourir l’ensemble des couples **(d, v)** de la base d’apprentissage. **Vérifier** que les 3 premiers couples de la base d’apprentissage génèrent les valeurs déterminées précédemment.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Base d’apprentissage** | |  | **Itération** | **v obtenu** | **Erreur** | **w3** | **b3** |
| D | v |  |  |  |  | **-10** | **5** |
| 0,1 | -17,6471 |  | 1 | 4 | -21,6471 | -10,10824 | 3,917645 |
| 0,15 | -13,2353 |  | 2 |  |  |  |  |
| 0,2 | -8,82353 |  | 3 |  |  |  |  |
| 0,3 | 0 |  | 4 |  |  |  |  |
| 0,5 | 17,64706 |  | 5 |  |  |  |  |
| … | … |  | … | … | … | … | … |

***Q22.*** A l’aide du tableau Excel, **procéder** aux copier-coller nécessaires afin de **déterminer** le nombre d’itérations total nécessaire sur la base d’apprentissage afin d’obtenir une erreur moyenne absolue sur les couples de la base d’apprentissage de moins **0,03**.

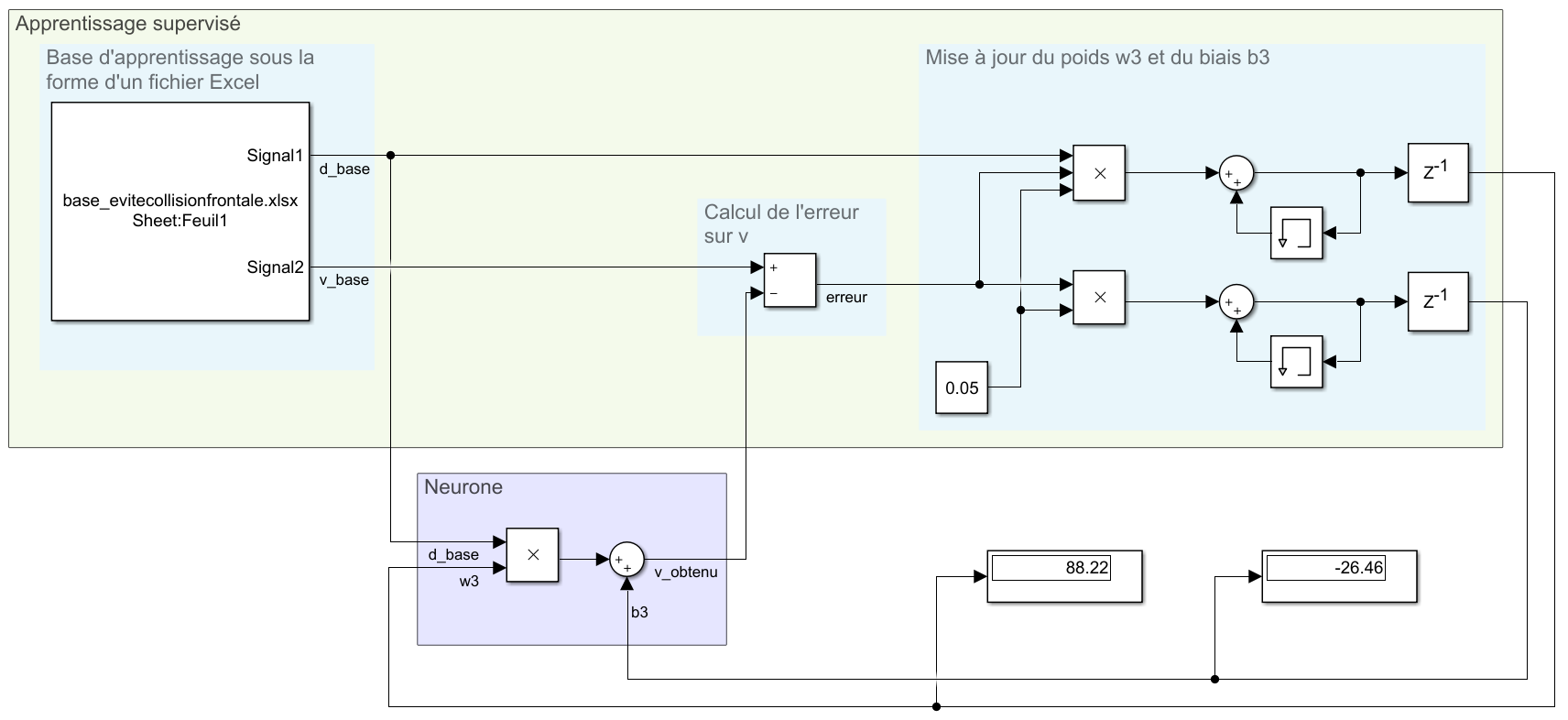
On souhaite modéliser sur Matlab / Simulink la phase d’**apprentissage supervisé** du perceptron à partir de la **base d’apprentissage** (ensemble des couples (d, v)).

***Q23.*** **Concevoir** un fichier Excel comportant les couples **(d, v)** de la base d’apprentissage. **Nommer** ce fichier **base\_evitecollisionfrontale.xlsx**



***Q24.*** **Ouvrir** Matlab / Simulink, **positionner**, **interconnecter** puis **paramétrer** les blocs indiqués ci-après. **Nommer** le fichier **evitecolisionfrontale\_apprentissage.slx**

Remarque : pour positionner un bloc il suffit de double-cliquer sur le fond blanc de la page Simulink puis de saisir le nom de bloc à positionner.



|  |  |
| --- | --- |
| Détail des blocs : | Paramétrage des blocs : |
| **From Spreadsheet**  Ce bloc constitue la **base d’apprentissage** décrite sous la forme du fichier Excel :  **Feuil1** : la feuille 1 est la feuille du fichier Excel qui comporte les couples (d, v).  **double** : les données **d** et **v** sont restituées dans Simulink comme des flottants à double précision.  **1** : l’itération sur la base d’apprentissage est réalisée chaque seconde. |  |
| **Product**  Réalise le produit des entrées. |  |
| **Sum**  Réalise la somme des entrées. | Rien à paramétrer ici (mais le nombre d’entrées peut être sélectionné dans les propriétés du bloc). |
| **Subtract**  Réalise la soustraction des entrées. | Rien à paramétrer ici (mais le nombre d’entrées peut être sélectionné dans les propriétés du bloc). |
| **Constant** |  |
| **Memory**  Ce bloc participe à la **mise à jour** du poids **w3** et du biais **b3** du perceptron.  En effet, il permet de **récupérer la valeur antérieure** (l’échantillon précédent souligné ci-dessous) du poids **w3** ou du biais **b3** en vue de le mettre à jour.  **w3 = w3 + …**  **b3 = b3 + …**  Il est donc ici nécessaire de définir une valeur initiale du poids **w3** et du biais **b3** du perceptron. Ces valeurs initiales seront nommées respectivement **w3\_initial** et **b3\_initial**.  Le point d’exclamation indique ici que la variable n’existe pas dans l’espace de travail de Matlab.  Pour créer ces variables dans Matlab, il suffit de saisir les commandes suivantes dans l’invite de commande de Matlab :  >> w3\_initial = -10  >> b3\_initial = 5  Si besoin, la modification de la valeur de ces variables est possible. |  |
| **Delay**  Sans rentrer dans des explications « superflues », ce bloc permet de prendre en considération les valeurs initiales de **w3** et **b3**. |  |
| **Display** | Pas de paramétrage à mettre en place si ce n’est un redimensionnement afin de visualiser toutes les valeurs affichées. |
| **Area color** | Disponible sur la bande latérale droite de la fenêtre Simulink afin d’ajouter des couleurs d’arrière-plan. |

***Q25.*** **Lancer** le modèle d’apprentissage sur 800 secondes (800 itérations sur la base d’apprentissage). **Relever** les valeurs finales du poids **w3** et du biais **b3** à l’issue des 800 itérations. **Ajouter** un bloc Display pour visualiser l’erreur. **Valider** la pertinence des données issues du tableur.

***Q26.*** **Renouveler** la simulation sur 2, 5, 10, 100, 1000 et 10000 itérations. **Etablir** un tableau selon l’exemple ci-dessous.

**Conclure** sur l’influence du nombre d’itérations dans l’obtention de valeurs pertinentes pour **w3** et **b3**.

**Expliquer** le compromis à faire à la vue de l’influence du nombre d’itérations sur le temps de calcul.

|  |  |  |  |
| --- | --- | --- | --- |
| **Nombre d’itérations** | **w3** | **b3** | **Erreur** |
| 2 |  |  |  |
| 5 |  |  |  |
| 10 |  |  |  |
| 100 |  |  |  |
| 1000 |  |  |  |
| 10000 |  |  |  |

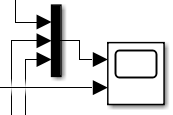
***Q27.*** **Renouveler** la simulation sur 800 itérations en modifiant la valeur du taux d’apprentissage **µ** à **0.05** en **0.01**, **0.06** puis **0.08**. **Etablir** un tableau selon l’exemple ci-dessous.

**Conclure** sur l’influence du taux d’apprentissage dans l’obtention de valeurs pertinentes pour **w3** et **b3** au bout d’un nombre d’itérations donné (ici 800).

**Expliquer** les conséquences d’une valeur **µ** trop faible ou trop élevée.

|  |  |  |  |
| --- | --- | --- | --- |
| **Constante µ** | **w3** | **b3** | **Erreur** |
| 0.01 |  |  |  |
| 0.05 |  |  |  |
| 0.06 |  |  |  |
| 0.5 |  |  |  |
| 1.1 |  |  |  |
| 1.5 |  |  |  |

***Q28.*** **Ajouter** un bloc **Scope** et des blocs **Mux** afin de visualiser l’évolution temporelle du poids **w3**, du biais **b3** et de l’**erreur**. **Faire** des captures d’écran pour **µ** = **0.05** et **µ** = **1.1**. **Commenter** les relevés temporels.



***b. Justification qualitative des relations mathématiques***

***Q29.*** **Lire** la justification qualitative des relations mathématiques utilisées durant la phase de rétro-propagation.

|  |
| --- |
| **Explication qualitative de la relation 1 :**  **w3 = w3 + µ × d × erreur** avec **µ** arbitrairement à 0,05 et **erreur = v – vobtenu**   * **w3** va voir sa valeur augmenter ou diminuer à chaque itération (en fonction du signe de l’erreur). * Si l’erreur est grande, **w3** est fortement corrigé. * Si la valeur de l’entrée **d** est grande, **w3** est d’autant plus corrigé (en effet si l’entrée **d** est fortement  activée, l’erreur engendrée provient du fait que la valeur de **w3** entraine une sur-réaction du neurone face à cette valeur importante de l’entrée). * **µ** est un coefficient (nommé *taux d’apprentissage*) qui permet une modification plus ou moins importante de **w3** à chaque itération (le choix de la valeur de **µ** est initialement arbitraire et ajustée si nécessaire). |

|  |
| --- |
| **Explication qualitative de la relation 2 :**  **b3 = b3 + µ × erreur** avec **µ** arbitrairement à 0,05 et **erreur = v – vobtenu**   * **b3** va voir sa valeur augmenter ou diminuer à chaque itération (en fonction du signe de l’erreur). * Si l’erreur est grande, **b3** est fortement corrigé. * Ici **b3** ne dépend pas de l’entrée **d** (tout se passe comme si **b3** était associé à une entrée de valeur constamment à 1). * **µ** est un coefficient (nommé *taux d’apprentissage*) qui permet une modification plus ou moins importante des de **b3** à chaque itération (le choix de la valeur de **µ** est initialement arbitraire et ajustée si nécessaire). |

***c. PROJET : Implémentation en Python de l’apprentissage supervisé sur la base robotique***

***Q30.*** **Compléter** le programme **apprentissage\_evitecollision.py** ci-dessous puis **tester** son fonctionnement avec 800 itérations sur la base d’apprentissage. **Etablir** le temps de calcul du poids w3 et du biais b3 par la PyBoard pour 5, 10, 100, 1000 et 10000 itérations sur la base d’apprentissage.

**Programme apprentissage\_evitecollision.py :**

**base\_apprentissage = [[0.1, -17.64],  
 [0.15, -13.23],  
 ...]  
  
def apprentissage\_supervisé(it, w3, b3, mu):  
 for i in range(it//len(base\_apprentissage)):  
 for ligne\_base in range(len(base\_apprentissage)):  
 # Propagation avant  
 ...  
   
 # Calcul de l'erreur  
 erreur = ...  
   
 # Rétro-propagation  
 w3 = ...  
 b3 = ...  
   
 return w3, b3  
  
w3\_initial = ...  
b3\_initial = ...  
mu = ...  
w3, b3 = apprentissage\_supervisé(..., ..., ..., ...)  
  
print(..., ...)**

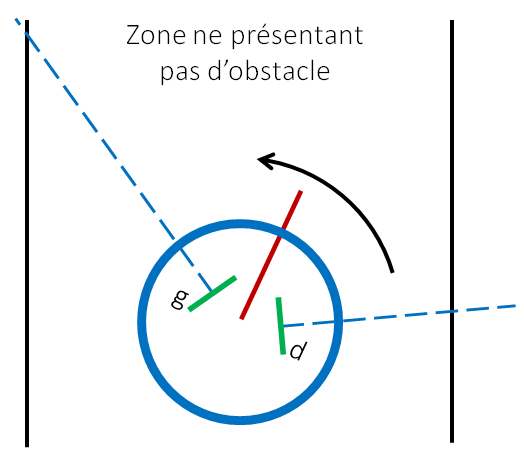
***d. PROJET : Implémentation en Python de l’apprentissage supervisé du perceptron***

***Q31.*** **Fusionner** le programme **apprentissage\_evitecollision.py** avec le programme **perceptron\_evitecollision.py** afin que les valeurs du poids **w3** et du biais **b3** obtenues à l’issue de l’apprentissage supervisé soient utilisées par le perceptron d’évitement de collision. **Enregistrer** le programme sous le nom **apprentissage\_perceptron\_evitecollision.py**. **Tester** le programme.

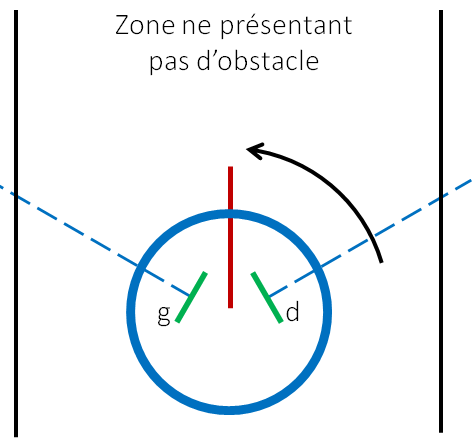
***IV. Mise en place d’un perceptron : positionnement angulaire***

***a. Modélisation***

On souhaite à présent que la base robotique mobile puisse **s’orienter angulairement** dans une direction ne présentant **pas d’obstacles**.



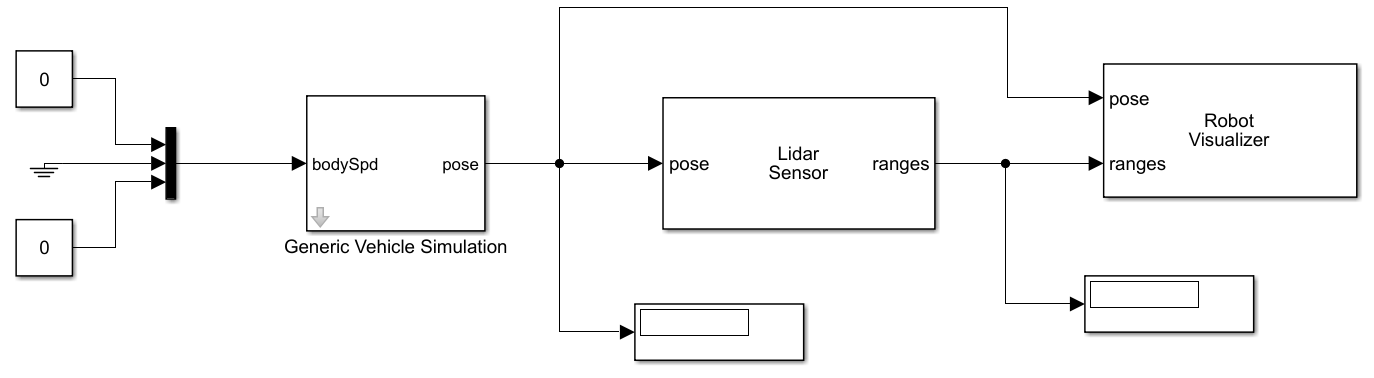
Pour cela on utilise à présent **deux capteurs de distance** gauche (**g**) et droit (**d**) positionnés à 60°.



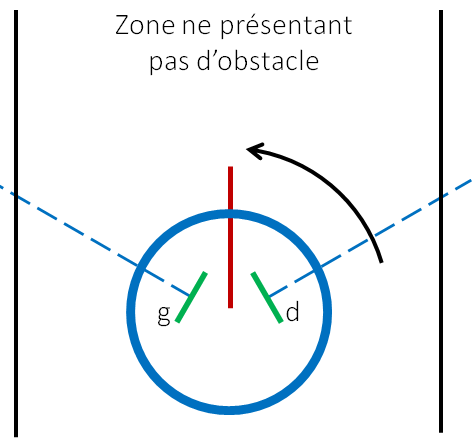
Grace à ces capteurs et aux moteurs, le robot mobile peut tourner sur lui-même afin de **se positionner angulairement** dans direction ne présentant pas d’obstacle.

***Q32.*** **Ouvrir** le modèle Simulink **evitecollisionfrontale.slx** conçu précédemment :

**Constante en rad/s**



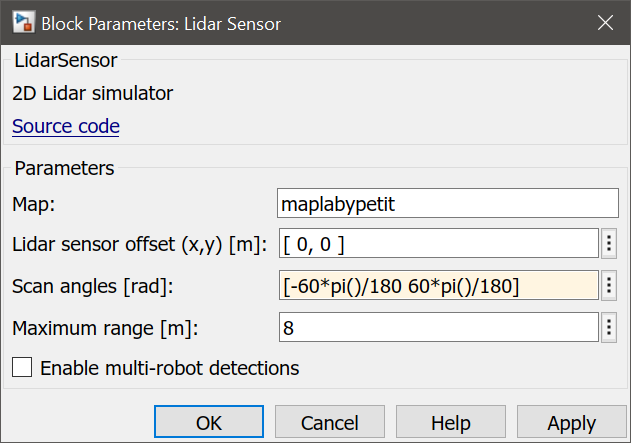
***Q33.*** **Modifier** la valeur de la constante à **0.01 rad/s** (avec une vitesse de translation de **0** m/s). **Relancer** la simulation (sur 10 secondes). Indiquer comment se comporte la base robotique. **Nommer** les liaisons en conséquence du modèle Simulink.



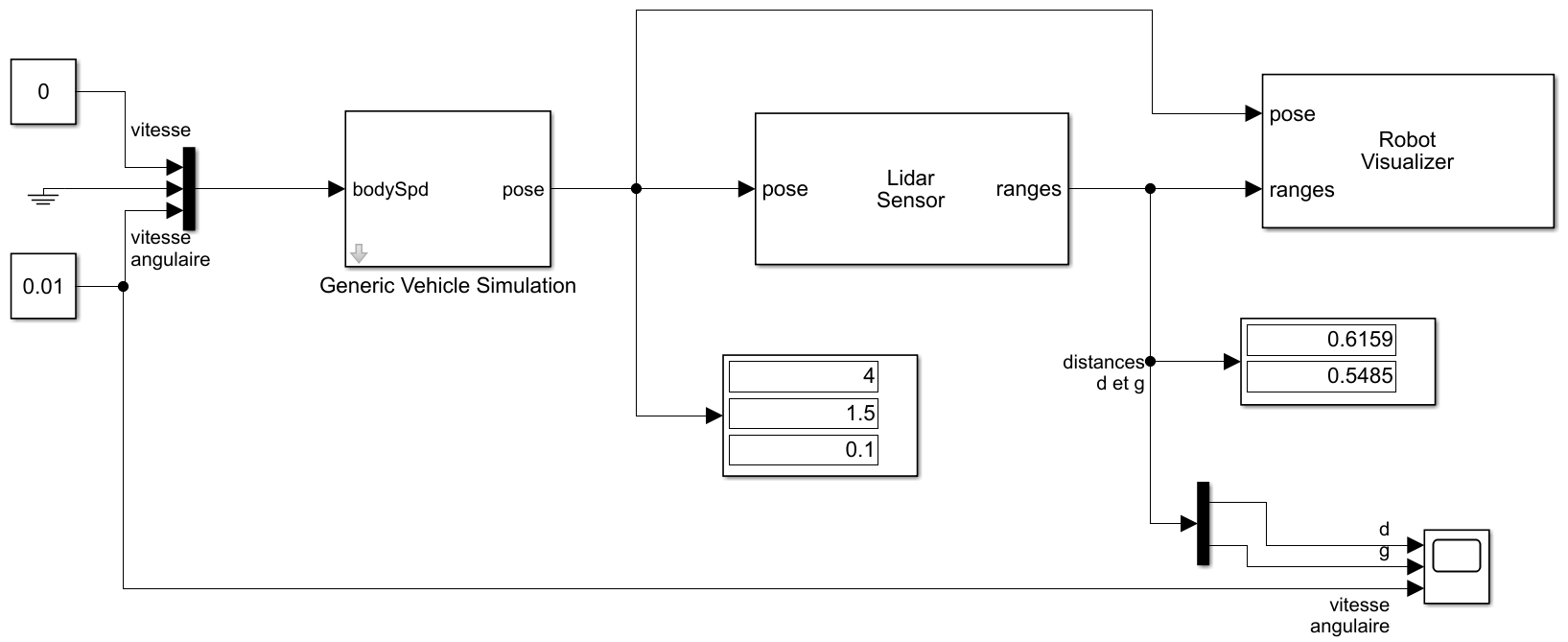
On va modifier **le nombre et la position angulaire des capteurs** présents sur le robot modélisé.

En effet, on suppose à présent que le robot dispose de deux capteurs de distance **disposés à -60° et +60°**.

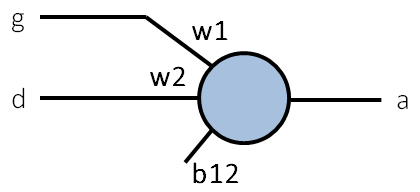
***Q34.*** **Modifier** les paramètres des blocs **Lidar Sensor** et **Robot Visualizer** afin d’ajouter les capteurs de distance disposer à **-60°** et **+60°** :



**Ajouter** un bloc **Demux** et un **Scope** afin de visualiser l’évolution temporelle des distances mesurées et de la vitesse angulaire.



On considère ici que les **distances mesurées** **g** et **d** par les capteurs de distance gauche et droit sont fournies en entrée d’un perceptron composé de :



* deux poids **w1** et **w2** ;
* et d’un biais **b12**.

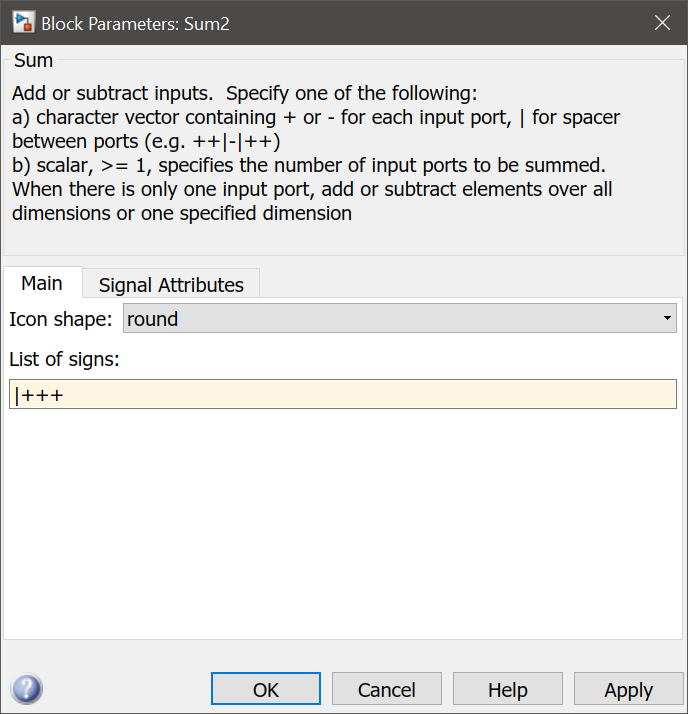
La sortie **a** du perceptron est **représentative de la** **vitesse angulaire** du robot (mouvement de rotation sur lui-même).

***Q35.*** **Donner** la relation liant la sortie **a** avec les entrées **g** et **d**.

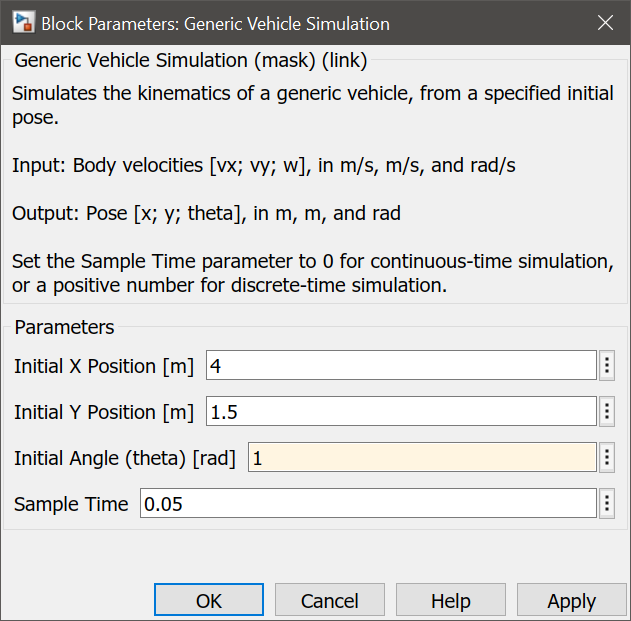
***Q36.*** En s’inspirant du modèle lié à l’évitement de collision, **mettre en place** dans le modèle Simulink précédent, le perceptron permettant le calcul de la vitesse angulaire du robot en fonction de la distance **g** et la distance **d**. **Nommer** le fichier **positionnementangulaire\_perceptron.slx**. **Lancer** le modèle. **Indiquer** le comportement du robot.

Remarques : la relation entre la valeur **a** représentative de la vitesse angulaire et la vitesse réelle sera considérée comme étant la suivante : **vitesse angulaire = a / 255**

l’ajout d’une entrée pour le bloc **Sum** est réalisé avec ce paramétrage :



***Q37.*** **Modifier** la position angulaire initiale du robot à **1 radian** (dans le bloc **Generic Vehicle Simulation**) puis **lancer** une simulation sur 40s. **Indiquer** le comportement du robot.



***b. PROJET : Implémentation en Python du perceptron de positionnement angulaire***

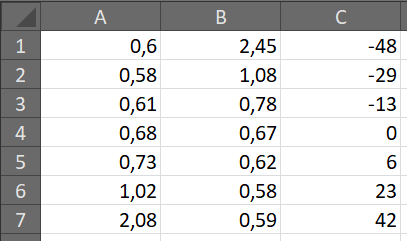
Cahier des charges : Le cahier des charges impose que le perceptron soit implanté en Python.

***Q38.*** **Proposer** et **mettre en place** une solution visant à implanter les capteurs de distance sur la base robotique.

***Q39.*** En vous inspirant du perceptron d’évitement de collision, **mettre en place** le perceptron de positionnement angulaire en Python sur la base robotique. **Enregistrer** le programme sous le nom **perceptron\_positionnementangulaire.py**. **Procéder** à des essais.

***c. Modèle d’apprentissage***

On souhaite modéliser sur Matlab / Simulink la phase d’**apprentissage supervisé** du perceptron de positionnement angulaire à partir de la **base d’apprentissage** constituée de l’ensemble, supposément pertinent (l’avenir nous le dira !), des couples **g** (colonne A), **d** (colonne B), **a** (colonne C) suivante :



***Q40.*** En s’inspirant du modèle d’apprentissage supervisé lié à l’évitement de collision, **concevoir** le modèle d’apprentissage supervisé du perceptron de positionnement angulaire à partir de la base d’apprentissage ci-dessus.

**Nommer** le fichier **positionnementangulaire\_apprentissage.slx**. **Procéder** à l’apprentissage et **donner** les valeurs des poids **w1**, **w2** et du biais **b12** à l’issue de l’apprentissage. **Relever** l’évolution de la valeur des poids et du biais, **discuter** du choix initial arbitraire des valeurs des poids et du biais.

Remarque :

- on prendra, arbitrairement, respectivement **-50**, **35** et **65** comme valeurs initiales de **w1**, **w2**, **b12** ;

- le taux d’apprentissage sera pris égal à **0.05** ;

- le nombre d’itération sera pris égal à **1000**.

***Q41.*** **Tester** la pertinence des valeurs des poids **w1**, **w2** et du biais **b12** obtenues à l’issue de l’apprentissage supervisé dans le modèle du perceptron de positionnement angulaire.

***Q42.*** Dans ce même modèle, **faire en sorte** que le robot avance à une vitesse constante de **0.1m/s**. **Décrire** le comportement du robot. **Indiquer** si la base d’apprentissage utilisée lors de l’apprentissage supervisé est pertinente.

***Q43.*** **Donner** le temps nécessaire pour parcourir un tour complet. **Augmenter** la vitesse de translation à **0.15** m/s puis **0.2** m/s. **Commenter**. **Proposer** une solution pour parcourir encore plus rapidement le parcours sans rentrer en collision avec une paroi. **Mettre en œuvre** cette solution dans le modèle. **Donner** le temps du parcours. **Relever** la vitesse translation maximale.

***d. PROJET : Implémentation en Python de l’apprentissage supervisé du perceptron***

***Q44.*** **Fusionner** les programmes **perceptron\_evitecollision.py** et **perceptron\_positionnementangulaire.py**. **Enregistrer** le programme sous le nom **perceptrons\_posang\_evitcol.py**. **Tester** le programme. **Evaluer** les écarts avec la modélisation.

***e. PROJET : Implémentation en Python d’un apprentissage supervisé in-situ.***

Cahier des charges : Le cahier des charges impose que l’apprentissage lié au positionnement angulaire soit réalisé in situ sur la base robotique mobile.

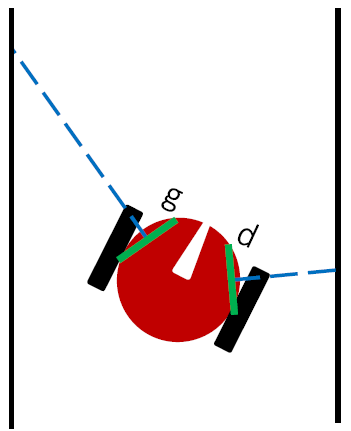
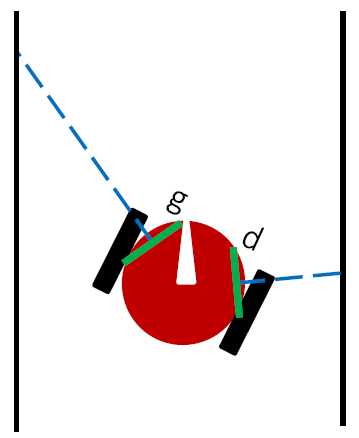
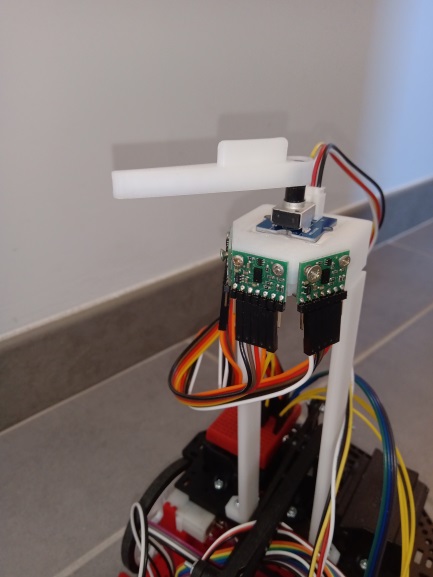
On souhaite à présent **créer la base d’apprentissage du positionnement angulaire in situ**, c’est-à-dire **en manipulant directement la base robotique**.

Ainsi la base robotique :

1. **procédera à la conception de la base d’apprentissage** liée au positionnement angulaire ;
2. puis **exécutera l’apprentissage supervisé** ;
3. et pour finir **utilisera les paramètres issus de l’apprentissage pour prédire le comportement à adopter** pour son positionnement angulaire.

***Q45.*** **Décrire** la solution qu’il peut être pertinent de mettre en œuvre. En cas difficultés, **lire** la suite de l’énoncé.

On se propose de **disposer** sur la base robotique **un potentiomètre dotée d’une aiguille** afin de **pointer la direction à atteindre** pour une position angulaire donnée :



*Exemple de solution possible*

***Q46.*** **Etablir** le schéma de câblage entre le potentiomètre et la PyBoard.

***Q47.*** **Compléter** le programme Python suivant afin d’**ajouter** un apprentissage supervisé in situ du perceptron de positionnement angulaire en Python au programme **perceptrons\_posang\_evitcol.py**. **Enregistrer** le programme sous le nom **apprentissage\_perceptrons\_posang\_evitcol.py**. **Tester** le programme. **Evaluer** les écarts avec la modélisation.

from pyb import Pin, Timer, ADC  
from machine import I2C, Pin  
import init\_vl53l0x  
import time  
  
**""" Base d’apprentissage initialement vide """**base\_apprentissage = []  
  
**""" Fonction d’apprentissage supervisé """**  
def apprentissage\_supervisé(it, w1, w2, b12, mu):  
 for i in range(it//len(base\_apprentissage)):  
 for ligne\_base in range(len(base\_apprentissage)):  
 # Propagation avant  
 a\_obtenu = ...  
   
 # Calcul de l'erreur  
 erreur = ...  
   
 # Rétropropagation des gradients d'erreur  
 w1 = ...  
 w2 = ...  
 b12 = ...  
   
 return w1, w2, b12  
  
**""" Initialisation des entrées / sorties """**  
adc\_X5 = ADC(Pin(**'Y12'**))  
  
i2c = I2C(scl=Pin(**'X9'**), sda=Pin(**'X10'**))  
  
capteur\_centre\_xshut = Pin(**'Y10'**, Pin.OUT)  
capteur\_gauche\_xshut = Pin(**'Y8'**, Pin.OUT)  
capteur\_droit\_xshut = Pin(**'Y7'**, Pin.OUT)  
  
tofl\_centre, tofl\_gauche, tofl\_droit = init\_vl53l0x.init\_capteurs\_centre\_gauche\_droite(i2c, capteur\_centre\_xshut, capteur\_gauche\_xshut, capteur\_droit\_xshut)  
  
tofl\_centre.start()  
tofl\_gauche.start()  
tofl\_droit.start()  
  
Y3\_pwm = Pin(**'Y3'**)  
Y4\_pwm = Pin(**'Y4'**)  
  
timer = Timer(4, freq=1000)  
Mot\_A\_Vitesse = timer.channel(3, Timer.PWM, pin=Y3\_pwm)  
Mot\_B\_Vitesse = timer.channel(4, Timer.PWM, pin=Y4\_pwm)  
Mot\_A\_Sens = Pin(**'Y5'**, Pin.OUT\_PP)  
Mot\_B\_Sens = Pin(**'Y6'**, Pin.OUT\_PP)  
  
Mot\_A\_Sens.high()  
Mot\_B\_Sens.high()  
Mot\_A\_Vitesse.pulse\_width\_percent(0)  
Mot\_B\_Vitesse.pulse\_width\_percent(0)  
  
**""" Construction de la base d'apprentissage """**for \_ in range(6):  
 Mot\_A\_Sens.high()  
 Mot\_B\_Sens.high()  
   
 for \_ in range(6):  
 Mot\_A\_Vitesse.pulse\_width\_percent(5)  
 Mot\_B\_Vitesse.pulse\_width\_percent(5)  
 time.sleep\_ms(150)  
 Mot\_A\_Vitesse.pulse\_width\_percent(0)  
 Mot\_B\_Vitesse.pulse\_width\_percent(0)  
 time.sleep\_ms(150)   
   
 donnee\_angle = adc\_X5.read()  
   
 angle = ...  
   
 distance\_gauche = tofl\_gauche.read()/1000  
 distance\_droit = tofl\_droit.read()/1000  
   
 print(**"Gauche : "** + str(distance\_gauche) + **" Droit : "** + str(distance\_droit) + **" Angle : "** + str(angle))  
   
 base\_apprentissage.append([distance\_gauche, distance\_droit, angle])   
  
 Mot\_A\_Vitesse.pulse\_width\_percent(5)  
 Mot\_B\_Vitesse.pulse\_width\_percent(5)  
 time.sleep\_ms(500)  
 Mot\_A\_Vitesse.pulse\_width\_percent(0)  
 Mot\_B\_Vitesse.pulse\_width\_percent(0)  
 time.sleep\_ms(0)  
  
**""" Initialisation arbitraire des poids et du biais """**  
w1\_initial = -50  
w2\_initial = 35  
b12\_initial = 65  
mu = 0.05  
  
**""" Apprentissage supervisé """**  
w1, w2, b12 = apprentissage\_supervisé(1000, w1\_initial, w2\_initial, b12\_initial, mu)  
  
print(w1, w2, b12)  
  
**""" Boucle principale """**  
while True:  
 distance\_gauche = tofl\_gauche.read()/1000  
 distance\_droit = tofl\_droit.read()/1000  
 print(**"Gauche : "** + str(distance\_gauche) + **" Droit : "** + str(distance\_droit))  
   
 a = ...  
  
 distance\_centre = tofl\_centre.read()/1000  
 print(**"Centre : "** + str(distance\_centre))  
   
 w3 = 88.235  
 b3 = 16.471  
 v = ...  
  
 print(a)  
   
 if a > 30:  
 a = 30  
 elif a < -30:  
 a = -30  
  
 print(v)  
   
 if v > 60:  
 v = 60  
 elif v < -60:  
 v = -60  
   
 if v >= 0:  
 Mot\_A\_Sens. ...  
 Mot\_B\_Sens. ...  
 else:  
 v = ...  
 Mot\_A\_Sens. ...  
 Mot\_B\_Sens. ...  
   
 compensation\_etat\_charge\_batterie = 2/3  
   
 Mot\_A\_Vitesse.pulse\_width\_percent(...  
 Mot\_B\_Vitesse.pulse\_width\_percent(...  
   
 time.sleep\_ms(10)

**Merci d’avoir lu jusqu’au bout !**

**Cette proposition de projet est évidemment encore perfectible, n’hésitez pas à la faire évoluer !**