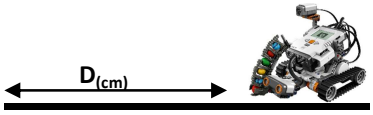
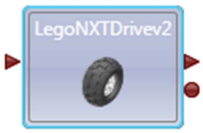
**CORRECTION DE LA SYNTHESE 2 NXT**

# 1- Contrôle de la distance de déplacement du robot

## 1.1 Contrôle de la distance de déplacement du robot




**Rappel** : Le déplacement du robot sur une distance $D_{(m)}$ peut être programmé avec un service « ***LegoNXTDrivev2*** » exécutant une action « *DriveDistance* ».
Cette action requière deux paramètres : « ***Distance*** » et « ***Power*** »




On donne ci-dessous un tableau rassemblant des mesures faites sur un des robots du lycée.
La puissance (**Power**) a été maintenue constante et égale à **70%.**

**Complétez le tableau ci-dessous.**

| Donnée placée dans le programme | « Distance » (m) | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Mesure** | **D (cm)** | 14,5 | 23,2 | 30,8 | 40,5 | 49,9 | 59,5 | 68,7 | 77 | 85,5 | 95 |
| **Ecart (cm) = « Distance » – D** | | **-4,5** | **-3,2** | **-0,8** | **-0,5** | **0,1** | **0,5** | **1,3** | **3** | **4,5** | **5** |

**Q1)** En utilisant les informations rassemblées dans ce tableau, **donnez** l'intervalle des valeurs du paramètre « ***Distance*** » pour lequel le robot se déplace sur une distance **D** en respectant une tolérance de ±1cm.
**Exprimez** cet intervalle en utilisant le signe ≤.

**0,3 ≤ Distance ≤ 0,6**

La grandeur maintenue constante est maintenant le **paramètre « Distance(m) » = 0.6**

**Complétez le tableau ci-dessous.**

| Donnée placée dans le programme | « Power » | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 |
|---|---|---|---|---|---|---|---|---|
| **Mesure** | **D (cm)** | 59 | 59.5 | 61.2 | 61,3 | 61,5 | 63 | 66 |
| **Ecart (cm) = « Distance » – D** | | 1 | 0,5 | -1,2 | -1,3 | -1,5 | -3 | -6 |

**Q2)** En utilisant les informations rassemblées dans ce tableau, **donnez** l'intervalle des valeurs du paramètre « ***Puissance*** » pour lequel le robot se déplace sur une distance **D** en respectant une tolérance de ±1cm.
**Exprimez** cet intervalle en utilisant le signe ≤.

**0,4 ≤ Puissance ≤ 0,5**

On donne en annexe la représentation graphique de « Distance » en fonction de « Power »

**Q3) Ecrivez** l'équation de « ***Power***» en fonction de « ***Distance***». **Précisez** les unités.

**Power = 113.Distance + 10 (Robot NXT2)**

**Q4)** Quelle est l'utilité de cette équation **? Expliquez** les opérations à effectuer pour vérifier sa validité.

Cette équation permet d'adapter la puissance appliquée sur chacune des roues du robot en fonction de la distance à parcourir.

Pour vérifier sa validité, il faut relever expérimentalement la distance parcourue d en fonction du paramètre « Puissance » et calculer l'écart **Ecart $_{(cm)}$ = « Distance » – D**.
Si -1 ≤ Ecart ≤ 1 sur l'intervalle 0,1≤ Distance ≤0,7 alors l'équation est valide (utilisable).

## 2 – Algorithmique

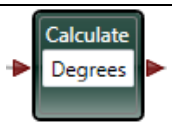
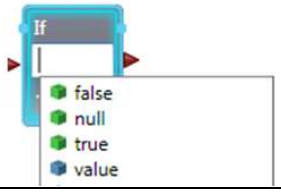
### 2.1 Mise en œuvre d'une boussole

Lors de ses déplacements, un robot doit effectuer des rotations. Pour cela, on l'équipe d'un capteur « boussole » capable de lui fournir un angle en degrés (Degrees). On se place dans le cas particulier ou le robot doit effectuer une rotation de + 90° (sens trigonométrique). On suppos e que le capteur délivre initialement la valeur 0°.

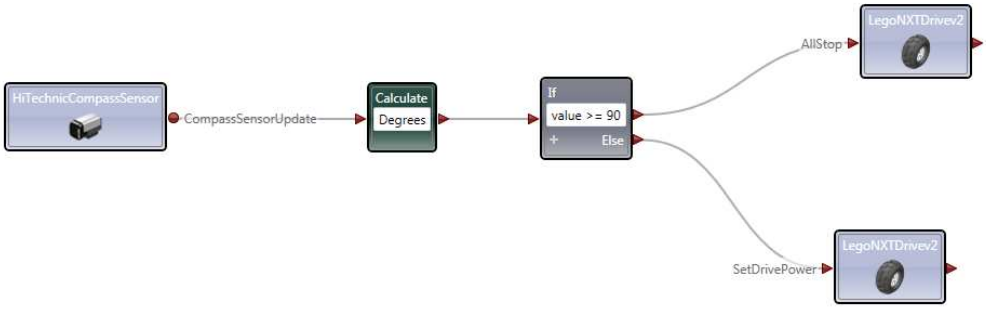
**Q5) Ecrivez** l'algorithme *Rotation_90* permettant pivoter d'un angle de +90°.

```
Algorithme
début
  Lire (Degree)
  si (Degrees ≥ 90) alors Arrêt du robot
                    sinon rotation du robot en sens anti-horaire
  fsi
fin
```

**Q6) Dessinez, à la règle,** le diagramme VPL correspondant à cet algorithme avec les blocs ci-dessous.

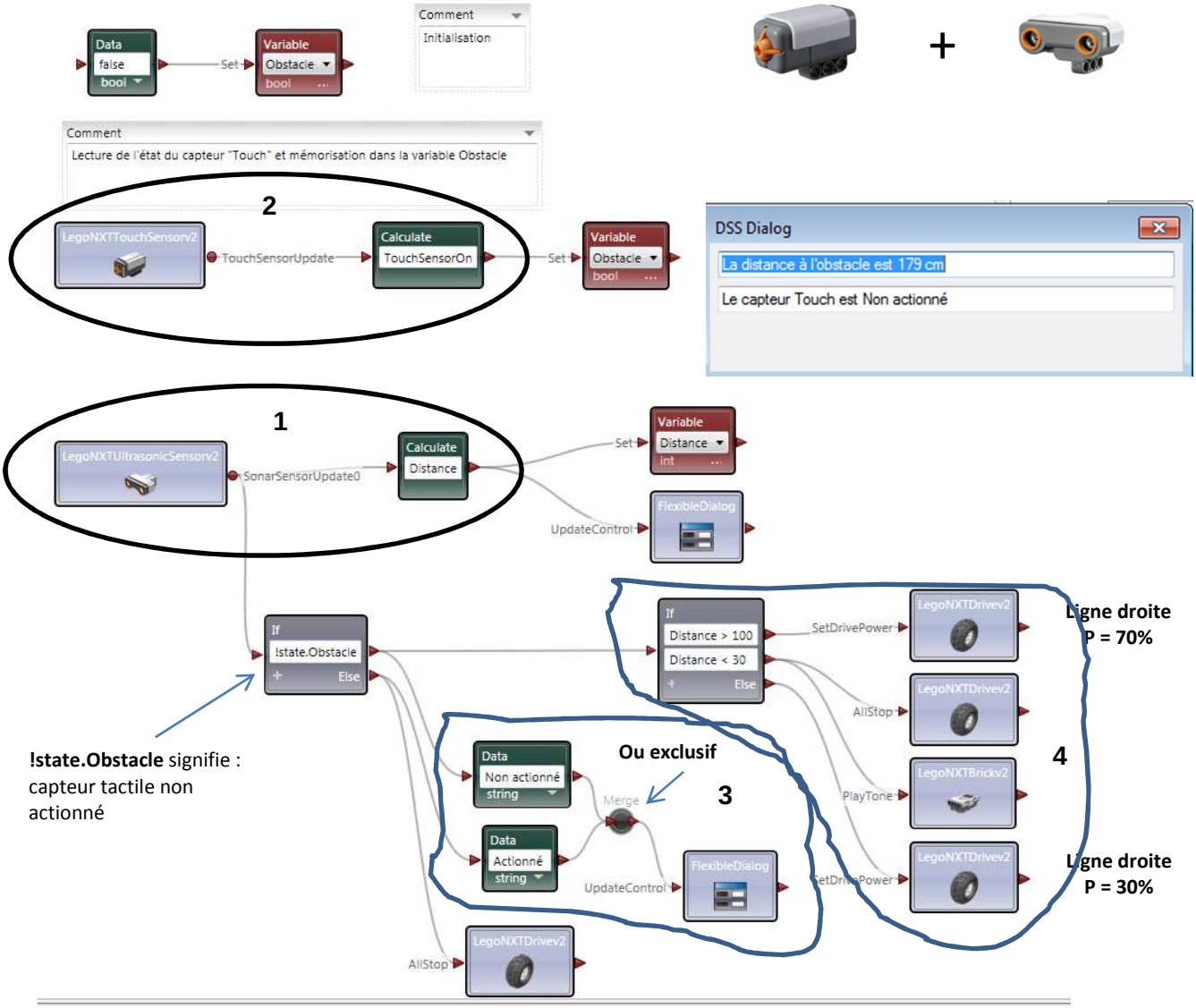
| **Activités** | **Services** | |
|---|---|---|
| | Fournit l'information « Degrees » issue d'un capteur « boussole » Lego | Commande les moteurs du robot. |
|  |  |  |
|  | *AnalogSensorUpdate*<br>*CompassSensorUpdate*<br>*ConnectionUpdate* | *AllStop*<br>*DriveDistance*<br>*EnableDrive*<br>*SetDrivePower* |
| **Operations possibles->** | | |

Dessin du diagramme



## 2.2 Détection d'un obstacle avec le capteur tactile et le capteur à ultrasons

On donne le diagramme VPL ci-dessous.




**!state.Obstacle** signifie : capteur tactile non actionné

**Q7) Entourez et numérotez :**

(1) La partie du diagramme correspondant à : ***Lire****(Distance)*

(2) La partie du diagramme correspondant à : ***Lire****(Capteur tactile)*

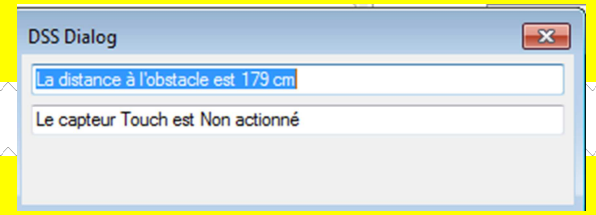
(3) La partie du diagramme correspondant à :

***si*** *(Distance>100cm)* ***alors*** *déplacer le robot en ligne droite (Puissance = 70%)*
***sinon si*** *(Distance < 30cm)* ***alors***
***début***
*Arrêt du robot*
*La brique Lego produit un son*
***fin***
***sinon*** *déplacement en ligne droite (Puissance = 30%)*
***fsi***
***fsi***

(4) La partie du programme correspondant à : ***Ecrire****(l'état du capteur tactile)*

**Q8) Ecrivez** l'algorithme *Capteur_Tactile* et l'algorithme *Déplacement* correspondant au programme de la question Q7).

```
Algorithme Capteur_Tactile
// Variable
   obstacle ← faux : booléen
début
   Lire(Capteur de choc)
   obstacle ← état logique du capteur de choc
fin
```




```
Algorithme Déplacement
// Variable
   Distance : entier

// On note d la distance mesurée entre le robot et l'obstacle

début
 Lire(d) ; Ecrire(d) ; Distance ← d
 si (le capteur tactile n'est pas actionné)
    alors
       début
         Ecrire (l'état du capteur tactile)
         si (Distance >100cm)  alors déplacer le robot en ligne droite (Puissance = 70%)
                               sinon si (Distance < 30cm)   alors
                                                               début
                                                                 Arrêt du robot
                                                                 La brique Lego produit un son (f = 1kHz - durée = 1s)
                                                                fin
                                                            sinon déplacement en ligne droite (Puissance = 30%)
                                                  fsi
         fsi
       fin
    sinon Arrêt du robot
 fsi
fin
```