UV 2.7 - TD/TE n°3 et 4 - Contrôle du robot DART

B. Zerr

Second Semestre 2016

1 Objectif du TE/TD

L'objectif de ces deux TD/BE est, en trois à quatre séances (9H à 12H), d'être capable de contrôler le robot DART et d'afficher sur une page WEB son état. Les premières séances se feront sur le robot DART virtuel et les suivantes utiliseront le robot réel. Ce travail s'effectue en trinôme (un robot par trinôme). Le résultat de ce travail servira de base aux épreuves de qualification du DART Challenge 2016. L'approche est progressive, en plusieurs étapes :

- étape 1 : fonctions de base du robot DART : commande des moteurs et acquisition des informations issues des capteurs,
- étape 2 : fonctions de déplacement du robot DART : ligne droite, rotation sur place, trajectoire circulaire, évitement des obstacles.
- étape 3 : reprise de la machine à états finis du TD n°1 (contrôle du robot NAO) et adaptation au contrôle du robot DART,
- étape 4 : passage au robot réel, adaptation des programmes, estimation des constantes (sens de rotation, position des sonars, ...)
- étape 5 : réalisation de la page WEB décrivant l'état du robot DART en utilisant le serveur WEB apache du robot.

Nous utiliserons le système d'exploitation Linux.

Remarque : pour le robot NAO, les étapes 1 et 2 ont été réalisées par la société Aldebaran Robotics. Lorsque l'on conçoit son propre robot (cas du robot DART), il faut écrire ces fonctions. Le "middleware" ROS (Robot Operating System) pourrait nous aider, mais c'est au programme de 2A !!! Nous allons donc écrire nous-mêmes ces fonctions.

Les robots DART sont maintenant (presque) opérationnels, voici les dernières infos :

- **quand la commande des moteurs est positive (ex. 60 60), le robot part en marche arrière !!!**
- **les sonars gauche et droit sont intervertis, cela est corrigible en modifiant l'ordre des sonars dans le tableur SONAR_NAMES du fichier sonar.py**
- **si la portée des sonars est trop faible, vous pouvez essayer d'augmenter la durée de l'impulsion émise sur Tx (en passant par exemple à une durée de 20 μs)**
- **le cap donné par la fonction getAngles() de l'IMU razor est quasi inutilisable. Il existe maintenant une procédure de calibration à réaliser avec un encadrant car il faut modifier le code C Arduino de l'IMU et le "téléverser" sur la carte RAZOR.**
- **après modification du code C Ardunio de la carte de contrôle moteur T-REX les encodeurs fonctionner**
- **les encodeurs arrières sont accessibles par le GPIO, un code Python permet d'y accéder (voir MOODLE)**

2 Installation du logiciel DART

Le logiciel DART permet l'exécution de votre programme sur les robots virtuels et réels. Lors de création d'un objet de la classe Daarrt (module Python dart.py), le programme teste si il est à bord du robot et, si ce n'est pas le cas, il lance le simulateur et exécute vos commandes sur le simulateur. Le logiciel DART comporte les éléments suivants :

- le module **dart.py** permettant de créer un objet robot (de la classe Daarrt), ce robot est soit virtuel, soit réel en fonction de l'endroit ou le code est exécuté : sur votre PC ou sur le robot.
- le dossier **vDaarrt** contenant le code de commande du robot virtuel et le code de commande du simulateur
- le dossier **drivers** contenant le code de commande du robot réel (vide pour l'instant)
- un dossier **gpio** avec les fichiers de commande des entrées-sorties numériques de la carte PCduino3
- un dossier avec le module **pylygon**,utile si ce module n'est pas installé sur vos PC

La partie graphique est réalisée à l'aide de pygame et les intersections (sonar, collisions) utilisent le module **pylygon**.

Télécharger de Moodle l'archive "dartsim-20160508.tgz", puis après décompression , exécuter la commande :

```
python dart.py
```

La pièce virtuelle doit s'afficher et le robot doit être immobile. Si le module **pylygon** n'est pas installé, une erreur se produit. Dans ce cas, faites l'installation en local du module **pylygon** sur votre compte à l'aide des commandes suivantes :

```
cd pylygon-1.3.2rc
python setup.py install --user
cd ..
```

La position initiale du robot DART dans le pièce virtuelle est définie par la position de la lettre D dans le fichier "./vDaarrt/data/World/world1.txt". Vous pouvez modifier ce fichier pour changer votre point de départ.

3 Fonctions de base du robot DART

Nous allons dans cette partie concevoir et implémenter les outils (fonctions Python) qui nous permettront d'accéder aux éléments matériels de base (capteurs et actionneurs) du robot DART.

3.1 Commande des moteurs

Les moteurs sont commandés par une carte de puissance pilotée par un microcontrôleur. Cette carte est reliée à la carte principale du robot (PCDuino3) par un bus I2C. Nous allons écrire 27 octets sur ce bus avec la fonction i2cWrite() pour commander les moteurs. La description de ces 27 octets est donnée dans la documentation de la carte T-Rex de commande des moteurs disponible sous MOODLE ou au lien suivant :

```
https://cdn.sparkfun.com/datasheets/Robotics/
T'REX%20robot%20controller%20instruction%20manual1.pdf
```

Le module **vTrex.py** du dossier **vDaarrt** fournit la fonction i2cWrite et utilise un dictionnaire (nommé package) pour définir ces 27 valeurs de commande.

Modifier la fonction **motor** de la classe **Daarrt** du module **dart.py** afin de commander les moteurs gauche et droit du robot DART. Le listing Python ci-après correspond à une vitesse nulle pour les deux moteurs :

```
# set right motor speed
self.trex.package['lm_speed_high_byte'] = 0
self.trex.package['lm_speed_low_byte'] = 0
# set right motor speed
self.trex.package['rm_speed_high_byte'] = 0
self.trex.package['rm_speed_low_byte'] = 0
# write the 27 command bytes (in package) to the i2C bus
self.trex.i2cWrite()
```

Vous pouvez insérer votre code de test de la fonction **motor** dans la partie "test unitaire" du module **dart.py** (voir exemple ci-dessous) ou créer votre propre programme de test.

```
# insert your test code here , example :
myDart = Daarrt()

# test motor commands
myDart.motor(50,-50)
time.sleep(2)

myDart.motor(-50,50)
time.sleep(2)

myDart.motor(50,50)
time.sleep(2)

myDart.motor(-50,-50)
time.sleep(2)


# end of simulation
myDart.ns.isAlive=False

# wait 1s to cleanly the end of simulation
time.sleep(1)
```

Tester l'avance en ligne droite et la rotation sur place.
Si le programme de simulation est en boucle infinie, vous pouvez l'arrêter en tapant "controle C".

3.2 Lecture de encodeurs

Les encodeurs permettent de définir le nombre de tours effectués par les roues du robot DART en comptant un certain nombre d'impulsions par tour de roues. Le robot virtuel produit, en principe, 200 impulsions

L'état de la carte de commande moteur est décrit par 24 octets. Ces 24 octets sont renvoyés par la fonction i2cRead et leur description est dans la documentation de la carte T-Rex de commande des moteurs disponible sous MOODLE ou au lien suivant :

`https://cdn.sparkfun.com/datasheets/Robotics/`
`T'REX%20robot%20controller%20instruction%20manual1.pdf`

Modifier la fonction **encoder** de la classe **Daarrt** du module **dart.py** afin qu'elle retourne les valeurs des encodeurs gauche et droit. Utiliser la fonction trex.i2cRead() et décoder le résultat avec la fonction struct.unpack() en utilisant la documentation de la carte T-Rex. Vous pouvez aussi

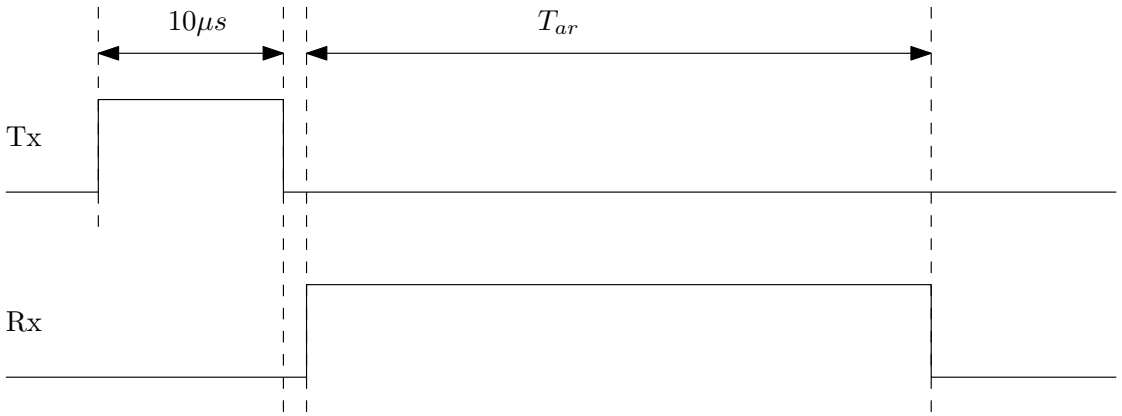

FIGURE 1 – Fonctionnement du sonar, une impulsion de courte durée (10 μs typiquement) est envoyée sur l'E/S Tx et le temps de trajet aller et retour est mesuré sur l'E/S Rx.

vous aider du code du simulateur en cherchant la fonction struct.pack() dans le code de la fonction trex.i2cRead().

Ecrire un programme de test qui fait tourner les moteurs pendant quelques secondes et qui relève les valeurs des encodeurs avant et après la rotation des moteurs.

3.3 Acquisition des sonars

Les sonars sont pilotés par les entrées-sorties (E/S) numériques de la carte PCDuino3. Chaque sonar utilise deux E/S numériques : une sortie numérique (Tx) servant à émettre l'impulsion ultrasonore et une entrée numérique (Rx) qui permet de détecter la réception de l'onde : elle passe à 1 lors de l'émission et passe à 0 lorsque l'onde réfléchie touche le capteur. Ce principe est illustré à la figure (fig .1).

La fonction **getSonars** de la classe **Daarrt** du module **dart.py** permet d'acquérir les 4 sonars dans l'ordre "back", "right", "front" et "left". Vous pouvez, si vous le souhaitez, ajouter une fonction par sonar (ex. **getFrontSonar** pour le sonar avant). Pour mesurer la distance parcourue, nous allons entrer dans le module vSonar.py et piloter les entrées-sorties de la carte PCDuino3. Le système linux du robot réel donne accès à l'entrée-sortie (E/S) numérique N à travers deux fichiers :

```
/sys/devices/virtual/misc/gpio/mode/gpioN
/sys/devices/virtual/misc/gpio/pin/gpioN
```

Sur le simulateur ces fichiers sont simplifiés :

```
gpio/mode/gpioN
gpio/pin/gpioN
```

Le premier fichier sert à définir si l'E/S numérique N est une entrée ou une sortie :

```
INPUT   = "0"
OUTPUT  = "1"
```

Le second permet d'écrire ou de lire une valeur logique sur l'E/S numérique N :

```
HIGH    = "1"
LOW     = "0"
```

Nous allons utiliser la classe **vGPIO** située dans le module **vSonar.py** pour définir les E/S de contrôle du sonar :

— la fonction **set** permet de définir si l'E/S est entrée ou en sortie
— les fonctions **read** et **write** permettre de lire et d'écrire sur es E/S numériques

Pour définir les numéros corrects d'E/S pour chaque sonar, nous utiliserons la table suivante :

sonar	n° Tx	n° Rx
Back	3	2
Right	5	4
Front	7	6
Left	9	8

Compléter la fonction **dist** de la classe **vSonar** dans le module **vSonar.py** afin qu'elle renvoie la distance entre le sonar et l'obstacle le plus proche.

Vérifier que la distance à l'obstacle diminue en avançant vers le mur.

3.4 Acquisition du cap

La mesure du cap du robot est réalisée par une centrale inertielle ou IMU (Inertial Measurement Unit). Nous utilisons une IMU RAZOR, dotée de beaucoup plus de fonctions que la simple mesure magnétique du cap :

`https://www.sparkfun.com/products/10736`

Cette IMU possède trois capteurs élémentaires :
— un gyroscope 3 axes (ITG-3200)
— un accéléromètre 3 axes (ADXL345)
— un magnétomètre 3 axes (HMC5883L)

et produit un vecteur de 9 valeurs. Ce composant assez complexe n'est pas simulé actuellement. Pour connaître le cap du robot, vous utiliserez la première des 3 valeurs de la fonction **getAngles** de la classe **vDaarrt**. Cette valeur est la recopie sans erreur de l'orientation du robot dans la pièce pygame.

Vérifier que le cap est correct (Nord orienté suivant l'axe des y) en faisant tourner le robot sur lui-même jusqu'à ce qu'il soit orienté au Nord.

4 Fonctions de déplacement du robot DART

L'idéal est de mettre toutes les fonctions de déplacement du robot DART dans un module python, par exemple **dartcmd.py**.

4.1 Orientation du robot selon un cap donné (méthode très simple)

Pour orienter le robot selon un cap prédéfini, nous allons utiliser la le cap donné par l'IMU RAZOR. Ouvrir un fichier Python (dartcmd.py par exemple) et créer la fonction **setHeadingSimple (head)** avec **head** la valeur de consigne en cap, c'est à dire le cap que doit atteindre le robot.

Pour faire tourner le robot sur lui-même et, ainsi modifier son cap, il faut faire tourner les moteurs gauche et droit selon des sens opposés. Les fonctions de haut niveau pour la commande des moteurs (fonction **motor** de la classe **vDaarrt** du module **dart.py**).
Un exemple d'algorithme pour la fonction **setHeadingSimple (head)** est présenté à la figure **Algorithm 1**.

Algorithm 1 Fonction - setHeadingSimple (head)

```
headOk ← False
Définir un sens opposé de rotation pour les 2 moteurs (pour que le robot tourne sur lui-même dans le sens horaire)
Définir la vitesse des moteurs (assez basse)
Appliquer vitesse et direction des moteurs au robot
while not headOk do
  headMes ← Mesure du cap
  headErr ← head − headMes
  if |headErr| < errMax then
    headOk ← True
    Mettre la vitesse des moteurs à zéro
  else
    Attendre un temps dt (typiquement 100 ms)
  end if
end while
```

Nous observons que la fonction **setHeadingSimple (head)** utilise l'erreur (**headErr**) entre la mesure du cap (**headMes**) et la consigne de cap (**head**) pour continuer ou interrompre la rotation du robot. Du fait des erreurs de mesure et des approximations de commande , il est en général impossible d'atteindre exactement le cap de consigne. Nous considérerons que le cap de consigne est atteint lorsque l'erreur entre le cap mesuré et la consigne passe sous un seuil prédéfini (**errMax**). Ce seuil peut être défini rigoureusement par un modèle d'erreur ou estimé empiriquement par essais successifs.

Note : nous appliquerons la seconde approche pour définir le seuil ***errMax****.*

4.2 Orientation du robot selon un cap donné (utilisation du signe de l'erreur)

La fonction **setHeading(head)** (voir **Algorithm 2**) est une évolution de la fonction précédente **setHeadingSimple(head)** afin d'obtenir une convergence plus rapide vers le cap de consigne. Le

sens de rotation n'est plus défini à priori mais va dépendre du signe de l'erreur. La rotation entre le cap actuel du robot et le cap de consigne sera ainsi de 180 degrés maximum. Par rapport à la méthode précédente, il faut changer le sens de rotation à l'intérieur de la boucle.

Note : attention, lorsque l'on change la sens de rotation d'un moteur, sa vitesse passe à zéro ; il faudra donc redéfinir la vitesse après chaque changement de rotation

Algorithm 2 Fonction - setHeading (head)

```
headOk ← False
Définir la vitesse des moteurs (assez basse)
while not headOk do
  headMes ← Mesure de la boussole en haute résolution
  headErr ← head − headMes
  ramener headErr dans l'intervalle [-180,+180] degrés
  if headErr >= 0.0 then
    Définir un sens opposé de rotation pour les 2 moteurs afin que le robot tourne sur lui-même dans le sens horaire
  else
    Définir un sens opposé de rotation pour les 2 moteurs afin que le robot tourne sur lui-même dans le sens anti-horaire
  end if
  if |headErr| < errMax then
    headOk ← True
    Mettre la vitesse des moteurs à zéro
  else
    Appliquer vitesse et direction des moteurs au robot
    Attendre un temps dt (typiquement 100 ms)
  end if
end while
```

4.3 Orientation du robot selon un cap donné par un correcteur proportionnel

La fonction **setHeadingProp(head,alpha)** est une évolution de la fonction précédente **setHeading(head)** utilisant à fois le signe de l'erreur et le niveau de l'erreur pour corriger le cap du robot.

Lorsque le robot est loin de son cap de consigne, la correction doit être rapide (peu précise). Lorsque le robot se rapproche du cap de consigne, la correction doit être plus précise (plus lente). Cette loi de commande est **"proportionnelle"** car elle corrige en utilisant une proportion de l'erreur. Cette proportion est définie par le coefficient **alpha** que vous définirez de façon empirique par essais successifs.

A l'intérieur de la boucle il faut donc à la fois modifier le sens de rotation des moteurs et leur vitesse de rotation.

Note : attention, lorsque l'on change la sens de rotation d'un moteur, sa vitesse passe à zéro ; il faudra donc redéfinir la vitesse après chaque changement de rotation

Note : les moteurs du robot possèdent une zone morte dont il faudra tenir compte. La loi de commande proportionnelle commande la vitesse de rotation avec une fonction de la forme vitesse = alpha |headErr|. Si l'erreur est très faible, la vitesse peut être dans la zone morte du moteur et l'on

préférera une loi de la forme vitesse = $vitesse_{MIN}$ + alpha $|headErr|$ avec $vitesse_{MIN}$ de l'ordre 50 (cela peut varier d'un robot à l'autre).

4.4 Régulation en ligne droite en utilisant les mesures issues de la boussole

En réutilisant les résultats précédents nous allons définir la fonction **goLineHeading (head, speed, duration)** qui permettra au robot de se déplacer pendant une durée donnée (**duration**) en ligne droite à une vitesse donnée (**speed**) et selon un cap donné (**head**).

La vitesse de chaque roue est la somme de la vitesse linéaire (**speed**) et des corrections de vitesse pour conserver le cap (**head**).

4.5 Régulation en ligne droite en utilisant les mesures issues des deux odomètres

Ecrire la fonction **goLineOdo (speed, duration)** qui permettra de faire la même chose que la fonction précédente, **goLineHeading (speed, duration)**, mais en remplaçant la méthode de mesure de l'erreur de cap. Au lieu d'utiliser la boussole, nous allons utiliser l'écart entre les valeurs lues sur les odomètres gauche et droit. Si cet écart est nul ou faible le robot avance en ligne droite. Cette fonction suppose que le robot est déja orienté au bon cap.

4.6 Evitement d'obstacles

Ecrire la fonction **obstcleAVoid()** permettant d'éviter les obstacles lorsque le robot se déplace en ligne droite.

4.7 Régulation suivant une trajectoire circulaire de rayon donné

Ecrire la fonction **goCurveOdo (speedTan, radius, sign, duration)** permettant au robot du suivre une trajectoire circulaire de rayon **radius** avec une vitesse linéaire (tangentielle) définie en m/s par **speedTan**. Le paramètre **sign** indique de quel coté le robot tourne (à gauche si **sign** vaut 1, à droite si **sign** vaut -1). Le contrôle de la trajectoire circulaire se fait à partir des mesures des odomètres en contrôlant la vitesse linéaire (tangentielle) de chaque roue du robot ; la différence des vitesses linéaire gauche et droite définissant la vitesse angulaire du robot.

Il est aussi possible d'utiliser la vitesse angulaire comme consigne en remplacement de la vitesse tangentielle ***speedTan****.*

4.8 Machine à états finis

Reprendre la machine à états qui nous a servi à piloter le robot humanoïde NAO et la modifier pour piloter le robot DART virtuel en utilisant les fonctions que vous avez définies précédemment.

5 Passage au robot DART réel

Dans cette partie, nous allons tester sur le robot réel, les programmes et fonctions écrites et testées sur le robot virtuel. 2016 correspondant à la première utilisation pédagogique du robot DART, il faut s'attendre à quelques surprises ! ? ! (voir en début de ce sujet de TD les dernières nouvelles)

Reprendre la solution de la partie 1 sous MOODLE afin de récupérer les modules "pilotes" (ou "drivers") du robot réel.

Le robot possède deux interrupteurs de mise en marche :

— un petit interrupteur à l'arrière gauche pour la carte principale PCDuino3

— un gros interrupteur rouge avec voyant au dessus du robot pour la partie puissance

Si votre robot part en "live" vous pouvez couper la partie puissance en appuyant sur l'interrupteur rouge. Il est conseillé d'avoir un petit programme qui juste l'arrêt des moteurs (ex stop.py fourni avec la solution MOODLE).

La connexion au robot réel se fait à travers un terminal à l'aide de la commande :

```
ssh uv27@172.20.25.1xx
```

le mot de passe est uv27 et xx correspond au numéro du robot de 01 à 18. Si tout se passe bien, vous êtes maintenant "dans le robot" dans le répertoire /home/uv27. La commande pwd doit indiquer /home/uv27 Afin de ne pas interférer avec les autres utilisateurs, créer votre propre répertoire :

```
mkdir grpXXXX
```

avec XXXX le pseudo personnalisé de votre groupe. Créer un second terminal et placez vous dans le répertoire ou se trouve votre script python de commande du robot DART (ex. dart.py). Pour copier vos fichiers sur le robot, utilisez la commande suivante :

```
scp -rp * uv27@172.20.25.1xx:grpXXXX
```

avec xx le numéro du robot et XXXX votre pseudo. Le mot de passe est uv27.

Repasser dans le premier terminal (i.e. sur le robot) et exécuter votre programme (ex dart.py) en tapant :

```
python dart.py
```

travail à réaliser Créer une machine à états finis permettant de télé-opérer le robot :

— arrêt (veille)

— marche avant et arrière

— rotation sur place (et éventuellement virage)

— arrêt sur obstacle (et éventuellement évitement d'obstacle)

FIGURE 2 – Page d'accueil principale du robot DART16

6 Interface WEB

6.1 Page d'accueil simple

L'interface WEB est outil pratique d'accès au robot et à ses principaux paramètres de fonctionnement. En tapant l'adresse IP de votre robot DART dans votre navigateur, les liens vers les différents groupes doivent apparaître (voir fig.2 pour le robot DART16 d'adresse IP 172.20.25.116). Vérifiez que cela fonctionne sur votre robot et, si ce n'est pas le cas faites appel à votre encadrant.

Plusieurs programmes et fichiers doivent être créés et placés sur le robot DART afin qu'il puisse fournir au navigateur internet sa page WEB dynamique.

Les deux emplacements de travail sur le robot DART sont les suivants :

— **/var/www/uv27/grpXXXX** : fichiers constituant la page WEB
— **/usr/lib/cgi-bin/uv27/grpXXXX** : fichiers CGI (scripts Python) exécutés lors de l'accès à la page WEB

avec **XXXX** le pseudo de votre groupe.

Rappel : Le mot de passe de l'utilisateur uv27 est uv27. Vous aurez besoin de ce mot de passe pour les commandes **scp** et **ssh**.

Rappel : Pour créer vos deux répertoires, ouvrez un terminal et tapez les commandes suivantes :

```
ssh uv27@172.20.25.116
cd /var/www/uv27
mkdir grpWeAreTheBest
cd /usr/lib/cgi-bin/uv27
mkdir grpWeAreTheBest
```

en supposant que le pseudo de votre groupe soit *WeAreTheBest*, ce dont je ne doute pas ...

6.2 Fichier principal index.html

Créer un nouveau fichier **index.html** et l'ouvrir en mode édition (éditeur de texte). Ce fichier sera votre page WEB principale sur le robot. A l'aide des balises html, title, body et p faire une simple page WEB avec un titre et deux paragraphes :

— le premier paragraphe souhaite la bienvenue sur le robot DARTx

— le second paragraphe affiche le pseudo de votre groupe et indique les membres du groupe.

Rappel : Pour envoyer votre fichier **index.html** sur le robot DART, tapez :

```
scp index.html uv27@172.20.25.1xx:/var/www/uv27/grpXXXX/
```

avec xx le numéro du robot DART et XXXX le pseudo de votre groupe.

Pour tester votre travail, accéder à votre robot DART à travers votre navigateur WEB (ex. Firefox) en tapant comme URL l'adresse IP du robot (172.20.25.1xx), avec xx le numéro du robot de 01 à 18. Le robot renvoie le contenu de sa page principale et le nom de votre groupe devrait apparaître. En cliquant sur le nom de votre groupe, votre page **index.html** doit ensuite s'afficher.

Note : le site w3school (lien http ://www.w3schools.com/html/default.asp) propose une introduction simple à l'usage de HTML

Note : vous pouvez si vous le souhaitez, completer la description de votre groupe par une image

6.3 Date et heure

Créer dans la page WEB un paragraphe avec la date et l'heure. Il faut pour cela utiliser la fonction python asctime() du module time. Comme le format HTML ne permet pas directement l'execution d'une fonction python nous procédons en deux étapes :

— étape 1 : un script CGI python (ex. datetime.cgi) permettant de récupérer et d'afficher la date et l'heure du robot sur sa page WEB

— étape 2 : créer une balise dans le fichier index.html permettant d'afficher le résultat de l'exécution du script CGI défini à l'étape 1

Un script CGI est un script python classique auquel on rajoute une partie initiale indiquant par quelle commande il doit être exécuté sur le robot DART et quel type de contenu il renvoie au navigateur. Il faut donc ajouter les deux lignes suivantes en début du fichier Python :

```
#!/usr/bin/python
print "Content-Type: text/html\n\n"
```

Il se peut que les accents posent problème,dans ce cas préciser le codage en utf8 :

```
#!/usr/bin/python
#coding: utf8
print "Content-Type: text/html\n\n"
```

*Note : L'insertion du script CGI dans le fichier **index.html** peut se faire par une balise de type < **object** > (voir description sur le lien suivant : http ://www.w3schools.com/tags/tag_object.asp*

Note : Le script sera déclaré dans le champ "data" de l'objet

Transférer les fichiers sur le robot et tester le bon fonctionnement à l'aide du navigateur WEB.

Rappel : copier avec **scp** les fichiers sur le robot DARTxx :

— **datetime.cgi** dans **/usr/lib/cgi-bin/uv27/grpXXXX**
— **index.html** modifié dans **/var/www/uv27/grpXXXX**
— utiliser le navigateur WEB à l'adresse 172.20.25.1xx (xx le numéro du robot, de 01 à 18)

Créer dans la page WEB principale (fichier **index.html**) un paragraphe avec la date et l'heure en utilisant un lien hypertexte pour le lancement du script CGI. Lorsque que l'on clique sur le texte du lien hypertexte, la date et l'heure apparaissent sur la même page ou sur une nouvelle page selon la configuration par défaut du navigateur.

*Note : L'insertion d'un lien hypertexte peut se faire par une balise de type <**a**> (voir description sur le lien suivant : http ://www.w3schools.com/tags/tag_a.asp*

Mettre à jour automatiquement la page WEB toutes les 10 secondes avec la balise < meta > en début de fichier , avant la balise <body> :

```
<meta http-equiv="Refresh" content="10; url=/uv27/grpXXX/index.html">
```

Vérifier que la date (l'heure) évolue toutes les 10 secondes.

6.4 Monitoring du robot DART

Les scripts CGI en python ne disposent pas des droits suffisants pour accéder directement aux capteurs du robot DART. Lorsque nous accédons au robot par le navigateur WEB nous sommes de simples utilisateurs anonymes et nous n'avons, en principe, pas accès aux fonctions de contrôle des capteurs.
Pour contourner ce problème, nous allons créer dans notre répertoire de travail **/home/uv27/grpXXXX** un programme de monitoring (exemple **/home/uv27/grpXXXX/dartstatus.py**) qui, toutes les cinq secondes :
— lit les informations des sonars et de la boussole (cap),
— enregistre ces informations dans le fichier texte (nommé par exemple **/usr/lib/cgi-bin/uv27/grpXXXX/dartstatus.txt**),

La structure du fichier **/usr/lib/cgi-bin/uv27/grpXXXX/dartstatus.txt**) est libre et, nous pouvons, par exemple adpoter un format en cinq lignes (4 sonars + cap) tel que :

```
frontSonar = 65.7
backSonar = 150.3
...
heading = 320.3
```

Vous pouvez ajouter d'autres informations (encodeurs, commande de vitesse des roues droites et gauches, niveau de batterie, etc.).

Création du script **/home/uv27/grpXXXX/dartstatus.py** et vérification sur le robot DART de la création automatique toutes les n secondes du fichier **/usr/lib/cgi-bin/uv27/grpXXXX/dartstatus**

Quelles sont les permissions sur fichier **/usr/lib/cgi-bin/uv27/grpXXXX/dartstatus.txt** (utiliser la ligne de commande : **ls -l** ou l'interface graphique) ?

Il faut donc donner au fichier **/usr/lib/cgi-bin/uv27/grpXXXX/dartstatus.txt** les permissions suivantes :

— lecture, écriture, exécution pour l'utilisateur
— lecture pour le groupe
— lecture pour les autres (utilisateurs anonymes quelquepart sur la planête

Cette modification devant se faire toutes les n secondes (à chaque écriture dans **/usr/lib/cgi-bin/uv27/grpXXXX/dartstatus.txt**), il serait fastidieux d'exécuter manuellement toutes les n secondes la commande :

chmod 744 /usr/lib/cgi-bin/uv27/grpXXXX/dartstatus.txt

Cette commande sera donc effectuée dans le script Python **/home/uv27/grpXXXX/dartstatus.py**.

*Note : la modifications des permissions d'un fichier peut se faire à l'aide la la fonction **os.chmod()** du module Python **os** et le module Python **stat**. Vous pouvez trouver les informations utiles en suivant les liens :*

— *http ://docs.python.org/2/library/os.html*
— *http ://docs.python.org/2/library/stat.html#module-stat*

*Note : la modifications des permissions d'un fichier peut se faire à l'aide la la fonction **os.chmod()** en utilisant une valeur numérique (ex 744) pour le mode. Cependant, attention !!!, 744 est une valeur définie en octal et il faut faire précéder cette valeur d'un zéro afin que Python comprenne qu'il s'agit bien d'une valeur en octal. Donc 0744 (octal) fonctionne et 744 (décimal) ne fonctionne pas (en décimal il faudrait écrire 484).*

Modifier le fichier **/home/uv27/grpXXXX/dartstatus.py** et vérifier que le fichier **/usr/lib/cgi-bin/uv27/grpXXXX/dartstatus.txt** possède les bonnes permissions

Création du script CGI python **/usr/lib/cgi-bin/uv27/grpXXXX/dartstate.cgi** qui va lire le fichier **/usr/lib/cgi-bin/uv27/grpXXXX/dartstatus.txt** et afficher sont contenu sur la page WEB.

Comme pour la date et l'heure, modifier votre fichier **index.html** pour que les résultats du script soient affichés. Vous pouvez utiliser l'une des deux méthodes :

— lien hypertexte avec la balise <**a**>
— objet graphique avec la base <**object**>

6.5 Monitoring graphique du robot DART

La partie graphique est réalisée en format SVG (Scalable Vector Graphics). Une présentation détaillée du format SVG et ainsi qu'un tutoriel d'initiation se trouvent à l'adresse suivante : http ://www.w3schools.com/svg/default.asp.

Insérer un graphique SVG dans votre fichier **index.html** à l'aide d'une balise et des éléments de référence du langage SVG.

Le formalisme est imposé. L'ensemble du graphique SVG se trouve à l'intérieur de la zone du

fichier délimitée par les balises <svg ...> et </svg> . Les éléments de référence et la version sont ajoutés dans la balise de début de la façon suivante :

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
```

Modifier le fichier **index.html** pour y insérer un graphique SVG constitué de :
— un cercle de rayon 40 centré en x=100, y=50
— un cercle de rayon 40 centré en x=100, y=50
— un rectangle de largeur 300 et de hauteur 100 centré en x= 250 et y=100
Pour le rectangle, la couleur de remplissage est constituée de 100 % de bleu, de 75 % de vert et de 75 % de rouge et la couleur de bordure est constituée à 25 % de vert, de 0% de rouge et de bleu. La largeur de la bordure est de 2 pixels. Les couleurs de remplissage et de bordure des cercles est libre.

Note : Vous pouvez vous aider du lien suivant pour mettre au point votre graphique SVG : http ://www.w3schools.com/svg/tryit.asp ?filename=trysvg_myfirst

Note : La couleur de remplissage est définie par fill=color et la couleur de bordure par stroke=color. Pour définir color on peut utiliser un codage hexadécimal avec trois octets (valeur de 0 à 255), l'un pour chaque couleur, exemples : rouge "#ff0000", vert "#00ff00", jaune "#ffff00", bleu "#0000ff". Il est aussi possible de définir color avec la fonction rgb(r,g,b) ou r,g, et b sont les proportions de rouge, vert et bleu (valeur entière entre 0 et 255) Pour les couleurs usuelles, il est aussi possible définir color littéralement, par exemple : "black", "red" , "blue".

Créer un fichier graphique SVG (exemple **mygraphic.svg**) et insérer le dans le fichier **index.html** à l'aide d'une balise de type < **embed** >.

Le fichier SVG posséde le format suivant :

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg xmlns="http://www.w3.org/2000/svg" version="1.1">

       inserer le code SVG ici

</svg>
```

Le lien http ://www.w3schools.com/tags/tag_embed.asp décrit l'utilisation de la balise < **embed** >.

6.6 Page d'accueil simple

En utilisant le cap et les 4 sonars, représenter dans un fichier graphique svg affichant en 2D la position des obstacles et leur orientation. Un obstacle peut être un marqueur (symbole) de coordonnées polaire (r_i,Θ_i), avec r_i la distance à l'obstace vu par le sonar i et Θ_i le cap du sonar i. La figure 3 montre un exemple d'affichage. Si on intégre le déplacement du robot, les affichages successifs doivent nous

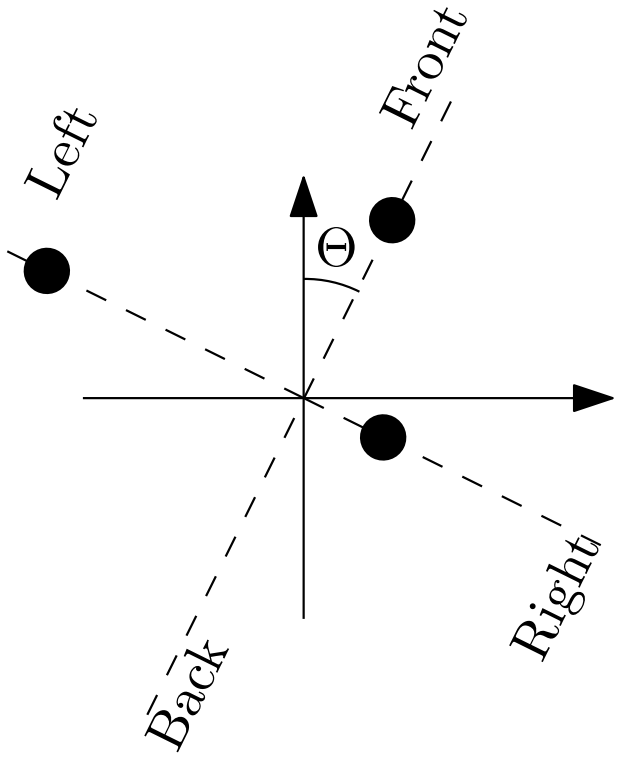

FIGURE 3 – Exemple de représentation graphique du cap et des obstacles vus par les sonars. Les détections des trois sonars avant, gauche et droite sont matérialisées par des disques noirs. Le sonar arrière ne détecte rien(pas de disque).

permettre de faire une carte d'obstacles (on devrait retrouver la forme carrée du parc à robot si tout va bien).

il faudra donc modifier le script CGI python **/usr/lib/cgi-bin/uv27/grpXXXX/dartstatus.py** afin qu'il crée automatiquement le fichier graphique SVG représentant les leds.