Poppy-Ergo-Jr DECOUVRIR VOTRE ROBOT

February 1, 2017

1 Découvrir votre Poppy Ergo Jr

Ce notebook qui permet à la fois d'insérer du code pour faire fonctionner le robot et de le commenter dans le même temps, va vous guider pour apprendre à programmer le Poppy Ergo Jr en Python.

1. Instancier votre robot
2. Accéder aux moteurs et les commander
3. Lire les valeurs des capteurs
4. S'initier à de la programmation de plus haut niveau

A partir de maintenant, votre robot Poppy Ergo Jr est connecté à l'aide du cable RJ 45 et votre caméra est aussi connectée.

```
In [ ]: %pylab inline

        from __future__ import print_function
```

% pylab inline est une commande python qui importe les modules *numpy* et *matplotlib*. L'option *inline* indique que les figures *Matplotlib* seront insérées dans le notebook lui-même plutôt que dans une fenêtre graphique séparée. C'est la seule option possible car votre système ne dispose pas d'interface graphique. Elle exécute en particulier les deux intructions * import numpy as np * import matplotlib.pyplot as plt

La dernière instruction importe du module *future* le sous module *print_function* qui permet de gérer les problèmes de compatibilité avec de futures versions de Python.

1.1 Instancier votre robot

Pour commencer à utiliser votre robot en Python, vous devez tout d'abord l'instancier. C'est le rôle du code suivant.

```
In [ ]: from poppy.creatures import PoppyErgoJr

        poppy = PoppyErgoJr()
```

Cela crée un objet Robot qui peut être utilisé pour accéder aux moteurs et aux capteurs. Il prend en chage toute la communication de bas niveau pour vous. Vous n'avez donc pas besoin de connaitre les détails du protocole de communication pour faire fonctionner un moteur. **Les champs *motors* et *sensors* du Robot sont automatiquement synchronisés pour correspondre à l'état de leurs équivalents matériels.**

1.1.1 QUESTIONS

1. Dans la ligne de code ci-dessus, quel est le rôle de poppy ?
2. PoppyErgoJr() est suivi de parenthèses que devez-vous comprendre ?

1.1.2 COMPLETER

1. Le code ci-dessous permet de mettre le robot ... dans la position appelée ...
2. Si le robot avait été nommé *poppy_bras* et la position avait été *position_leve*, quelle instruction auriez-vous du envoyer au robot pour lui demander de se mettre dans cette position ?

```
In [ ]: poppy.rest_posture.start()
```

1.2 Accéder aux moteurs

Dans un Poppy Ergo Jr, le moteur est défini comme illustré ci-dessous : **Image à rajouter**

A partir de l'objet Robot, vous pouvez récupérer directement la liste de tous les moteurs conectés à l'aide de l'instruction construite de la manière suivante : nom_du_conteneur_de_l_objet_robot.motors

1.2.1 QUESTIONS

1. Quel est le nom du conteneur/variable qui représente votre robot ?
2. L'instruction ci-dessous est-elle valide ? Si non, la corriger.

```
In [ ]: poppy_ergo_jr.motors
```

Comme vous pouvez le constater, vous obtenez une liste de tous les moteurs de votre objet Robot.

Vous pouvez alors récupérer tous les noms des moteurs

En Python, une syntaxe possible de la boucle *pour* est la suivante :

for *itérateur* **in** *une_liste* ** : **

```
*corps de la boucle pour*
```

1. Qui est le "l'itérateur" dans la boucle suivante ? Décrivez son fonctionnement.
2. Quelle est la liste ? Ecrivez-là de manière explicite.

```
In [ ]: for m in poppy.motors:
            print(m.name)
        #print ("terminé")
```

TRAVAIL :

- Si dans la dernière ligne, on supprime le caractère #, l'instruction print("terminé") sera exécutée. Combien de fois sera affichée la chaine de caractères "terminé" ?

Les instructions répétées dans la boucle doivent avoir la même indentation ou être dans une structure qui est au même niveau d'indentation.

- Faites en sorte que "terminé" soit affiché à la suite du nom de chaque moteur.

On peut aussi appeler chacun des moteurs par son nom (m1,m2,etc.) à l'aide l'instruction **contenur_du_robot.nom_du_moteur** comme dans l'exemple ci-dessous.

```
In [ ]: poppy.m1
```

1.3 Lire les valeurs des moteurs

A partir de l'objet moteur vous pouvez accéder à ses divers registres. Les principaux sont :

- **present_position**: retourne la position courante du moteur en degrés
- **present_speed**: la vitesse courante du moteur en degrés par seconde
- **present_load**: la charge de travail du moteur (en pourcentage de la charge maximale)
- **present_temperature**: la température du moteur en degrés celsius
- **angle_limit**: les limites atteignables du moteur (en degrés)

Ils sont accessibles directement.
Syntaxe possible : **nom_du_robot.nom_du_moteur.registre**

1.3.1 QUESTION

Dans l'instruction suivante, quel registre de quel moteur de quel robot est demandé ? Que doit retourner cette instruction ?

```
In [ ]: poppy.m1.present_position
```

On peut avoir la position courante de tous les moteurs à l'aide de l'instruction ci-dessous.

1.3.2 QUESTIONS

1. Pourquoui a-t-on des crochets ?
2. Quelles sont toutes les valeurs que va prendre l'itérateur *m* ?
3. Quelle est la structure de poppy.motors ?

```
In [ ]: [m.present_position for m in poppy.motors]
```

Il est important de comprendre que **"poppy.m1.present_position" est automatiquement mis à jour** avec la position courante du moteur réel (à 50Hz). Comme pour les autres registres, la fréquence de mise à jour n'est pas la même et dépend de son importance. Par exemple, la température est rafraichie à 1Hz puisqu'elle ne varie pas très vite.

1.4 Remarque

Lorsque vous appuyez sur la touche tabulation après avoir mis un **.**, vous pouvez voir la liste des registres pour un objet. Essayez dans la cellule de code suivante et observez.

Vous n'avez donc pas besoin de retenir les divers registres existants ni leurs noms.

```
In [ ]: poppy.
```

1.5 Commander les moteurs

Au début des registres présentés précédemment, il y en a d'autres utilisés pour envoyer des instructions/commandes au robot. Par exemple, la position des moteurs est séparée en deux registres distincts.

- En lecture seulement **present_position** du moteur
- En lecture et écriture **goal_position** qui envoie au moteur une position cible qu'il va essayer d'atteindre.

Pour mettre un moteur dans une nouvelle position, vous pouvez écrire :

```
In [ ]: poppy.m1.goal_position = 20
```

1.5.1 QUESTIONS

1. Quel effet va avoir l'intruction ci-dessus ?
2. En quelle unité est exprimé 20 ?
3. Comment faire pour que le moteur tourne dans l'autre sens de la même manière ?

```
In [ ]: #COMPLETER
        poppy.m1.goal_position =
```

Dans ces exemples, le moteur tourne aussi vite que possible (c'est le fonctionnement par défaut). Vous pouvez changer la vitesse maximale du moteur qui est enregistrée dans le registre moving_speed du moteur :

```
In [ ]: #COMPLETER : pour mettre le registre moving_speed du moteur m1 de votre rob
        poppy.                =50
```

Maintenant le moteur *m1* ne peut pas aller plus vite que 50 degrés par seconde. Faites le se déplacer une nouvelle fois pour constater la différence.

```
In [ ]: poppy.m1.goal_position = 90
```

Les principaux registres sont :

- **goal_position** : position cible en degrés
- **moving_speed** : la vitesse maximale atteignable en degrés par seconde
- **compliant** : expliqué ci-après

Les servo moteurs dynamixel ont deux modes :

- **stiff** : le mode normal des moteurs dans lequel ils peuvent être controlés.
- **compliant** : un mode dans lequel les moteurs peuvent être bougés librement à la main. Ce mode est particulièrement utile lors d'interactions physiques humain-robot.

Vous pouvez les faire passer d'un mode à l'autre en utilisant le registre *compliant*. Ce registre ne pouvant être associé qu'à deux valeurs, c'est un booléen qui ne peut prendre que les valeurs *True* et *False*. Vous pouvez par exemple mettre le moteur *m6* en mode compliant via :

```
In [ ]: poppy.m6.compliant = True
```

Vous pouvez normalement faire bouger ce moteur à la main. Par exemple, cela vous sera utile pour programmer votre robot par démonstration (cf. notebook correspondant).

Et le remettre en mode *stiff* :

```
In [ ]: poppy.m6.compliant = False
```

1.6 Contrôler les LED des moteurs

Les moteurs XL-320 motors utilisés dans le Poppy Ergo Jr ont un petite LED de couleur. Vous pouvez changer sa couleur en utilisant *pypot*. C'est utile pour rendre votre robot plus vivant, customisé ... Chaque moteur possède un registre *led* auquel on peut affecter un couleur *'green'*, *'red'* parmi les couleurs disponibles.

1.6.1 QUESTION

Comment faire pour que la *led* du moteur *m3* du robot *poppy* soit verte ?

```
In [ ]:
```

En affectant la valeur *'off'* à ce registre on éteint cette led.

1.6.2 QUESTION

Eteindre la *led* du moteur *m3* du robot *poppy*.

```
In [ ]:
```

1.6.3 QUESTION

1. Avant de le lancer, pouvez-vous décrire l'effet de ce jeu d'instructions ?
2. Comment faire pour allumer une led sur deux en rouge et une led sur deux en vert dans cet ordre ? En Python la syntaxe du *si ... alors ... sinon ...* est la suivante :

if (condition) :

```
instructions du alors
```

else :

```
instructions du sinon
```

suite des instructions du programme ...
C'est l'indentation qui structure le programme.

```
In [ ]: import time

        for m in poppy.motors:
            time.sleep(0.5)
            m.led = 'yellow'
            time.sleep(1.0)
            m.led = 'off'
```

Vous pouvez connaitre toutes les couleurs disponibles à l'aide la commande suivante :

```
In [ ]: from pypot.dynamixel.conversion import XL320LEDColors

        print(list(XL320LEDColors))
```

1.7 Lire des capteurs/sensors

```
In [ ]: import cv2
        # pour utiliser la caméra
        %matplotlib inline
        import matplotlib.pyplot as plt
        from hampy import detect_markers
        # pour détecter des marqueurs

        img = poppy.camera.frame
        plt.imshow(img)
        # Que contient le conteneur img ?
        # Que signifie plt.imshow(img) par quelle autre commande peut-on le remplac
```

Lire des capteurs se fait exactement de la même manière que lire des registres de votre robot.
Vous pouvez accéder à vos capteurs via :

```
In [ ]: poppy.sensors
```

Ici, nous avons 2 capteurs : * **une caméra** * **un détecteur de marqueur**
Vous pouvez y accéder via leur nom :

```
In [ ]: poppy.camera
```

Vous pouvez récupérer tous les registres existants d'un capteur :

```
In [ ]: poppy.camera.registers
```

et recupérer et afficher une image à partir de la caméra :
Les deux premières lignes suivantes permettent d'importer la bibliothèque qui gère l'image.
img est un conteneur qui contient l'image récupérée par la caméra du poppy.
La ligne 4 permet d'afficher l'image contenue dans le conteneur *img*.

```
In [ ]: %matplotlib inline
        import matplotlib.pyplot as plt
        img = poppy.camera.frame
        plt.imshow(img)
```

Comme pour les moteurs, les valeurs des capteurs sont automatiquement synchronisées en arrière plan avec le capteur physique. Si vous exécutez une nouvelle fois les instructions précédentes, vous obtiendrez une image plus récente.

```
In [ ]: plt.imshow(poppy.camera.frame)
```

1.8 Comportements de haut niveau

Le robot Poppy Ergo Jr intègre un jeu de comportements prédéfinis. Il peut s'agir de postures spécifiques comme la posture de repos - rest_posture utilisée au début - ou d'une danse, ...

Vous pouvez en trouver la liste exhaustive en utilisant l'instruction suivante qui liste les *primitives* ou *comportements* :

```
In [ ]: [p.name for p in poppy.primitives]
```

Ces comportements (ou primitives en "terminologie poppy") peuvent être démarrés, arrêtés, mis en pause, etc.

```
In [ ]: poppy.tetris_posture.start()
```

Vous pouvez faire danser le Poppy Ergo Jr pendant 10 secondes :

```
In [ ]: import time

        poppy.dance.start()
        time.sleep(10)
        poppy.dance.stop()
```

1.9 Pour aller plus loin

Maintenant que vous avez appris les bases de ce que vous pouvez faire avec un Poppy Ergo Jr, il y a encore plus à découvrir :

- Comment enregistrer et rejouer un mouvement par démonstration ?
- Comment définir un comportement de haut niveau (utiliser des marqueurs visuels associés à des actions) ?
- Comment utiliser Poppy Ergo Jr comme objet connecté et le faire communiquer avec le reste du monde en utilisant des requêtes http ?
- ...

Vous pouvez trouver d'autres exemples dans la doc ou dans le dossier notebook suivant.

```
In [ ]:
```