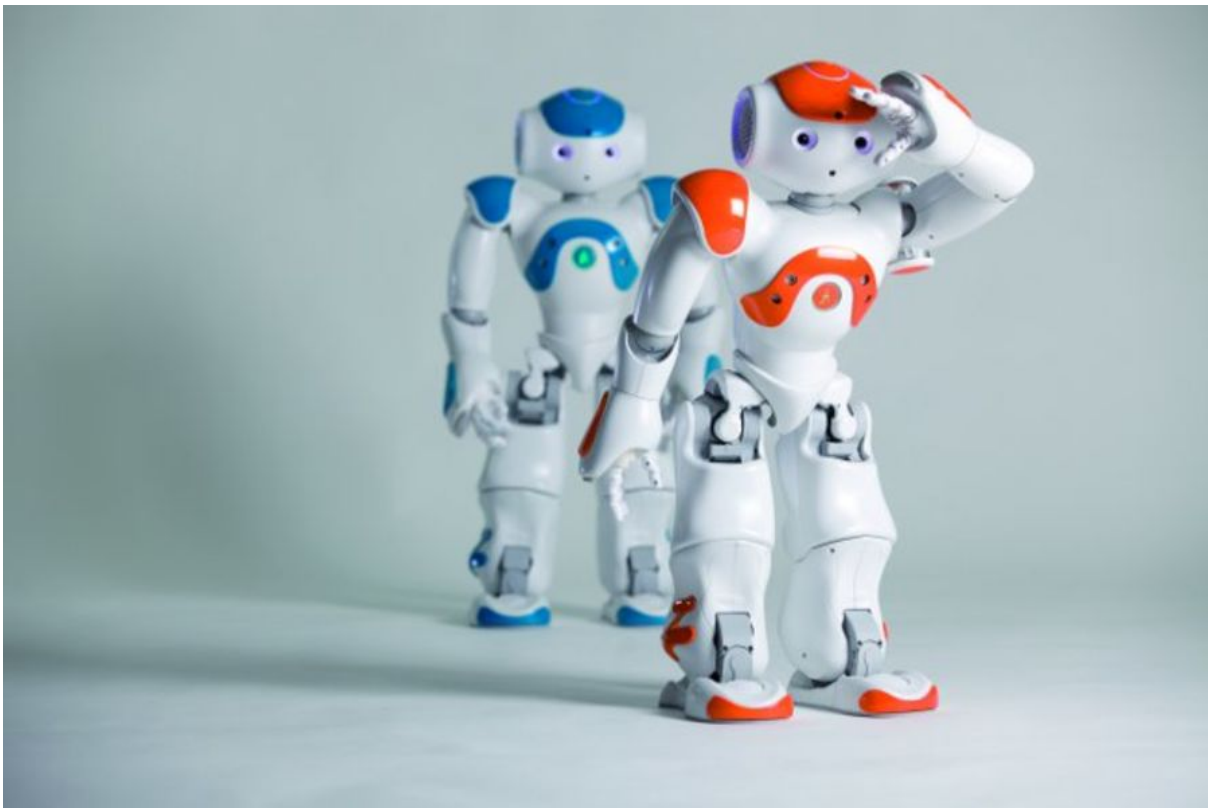
MCMANUS Ronan L1 - CMI - Info
Tuteur : CARMAGNAC Christophe

Robot Nao



Sommaire

1/ Présentation du robot

Le robot NAO, créé par Aldebaran Robotics (aujourd'hui SoftBank Robotics), est un robot humanoïde dont l'histoire remonte aux années 2000. Il a été conçu par Bruno Maisonnier, fondateur de la société, dans le but de développer un robot interactif et polyvalent.

Les premières versions de NAO ont été introduites en 2006. Le robot se distingue par sa forme humanoïde, mesurant environ 58 centimètres de hauteur et pesant environ 5 kilogrammes.

Équipé de divers capteurs tels que des caméras, des microphones, des capteurs de pression et de collision, NAO est capable d'interagir avec son environnement, de reconnaître les objets et les visages. Ses articulations motorisées lui permettent de marcher, de se tenir debout et d'effectuer différentes actions.

NAO est contrôlé par un logiciel embarqué qui lui permet d'exécuter des tâches programmées, de reconnaître la parole et les visages des utilisateurs, ainsi que de se connecter à Internet pour accéder à des informations et des services en ligne.

Ce robot a connu un grand succès dans le domaine de la recherche en robotique et a été largement utilisé dans les laboratoires et les universités du monde entier. Il a également été utilisé dans l'éducation, notamment pour l'enseignement de la programmation et des sciences aux étudiants qui est le cas pour ce projet.

Au fil des années, plusieurs versions de NAO ont été développées, améliorant ses performances, son autonomie et ses capacités. Il a également été employé dans des applications commerciales telles que la santé, l'assistance aux personnes âgées et l'animation.

NAO a joué un rôle majeur dans la popularisation de la robotique humanoïde et a ouvert la voie à d'autres robots similaires tels que Pepper, également développé par SoftBank Robotics.

2/ L’objectif du projet

L'objectif du projet est de déplacer le robot NAO d'un point A à un point B tout en évitant un obstacle. Cela signifie que le robot doit être capable de comprendre ce qui l'entoure, quand on lui donne une longueur et de contrôler ses mouvements.

Pour y parvenir, NAO va utiliser ses sonars pour détecter à quelle distance il se situe de l’objet. Ces capteurs lui permettent de comprendre où se trouve l'obstacle et comment s'orienter par rapport à lui.

L’objectif n’a pas pu être atteint suite à des problèmes donc le robot se contentera de contourner un obstacle pas trop large sans revenir sur sa route. On verra par la suite comment ceci a été implémenté[1].

3/ Rendu finale (vidéo)

https://youtube.com/shorts/gSRI5c3SgBY

Dans cette vidéo, on aperçoit le robot en biais, la cause pour cela, on suppose, que le robot a un souci à sa jambe gauche ce qui le fait dévier sur la gauche, c’est pour ça qu’il est dans cette position.

Donc, dans un premier temps, on lui donne une certaine distance à avancer et quand il rencontre l’obstacle, il s’arrête se mets dans une position par défaut, ensuite, il s’avance par la droite, car il se rend compte que l’objet est plus à sa droite qu’à sa gauche enfin, il s’arrête et recommence à avancer tout droit.

4/ Prise en main de l'api 1.14.5 de nao

Les premières semaines du projet, j'ai essayé de prendre en main en créant des fonctions simples et en mettant en place les différents proxy de NAO pour les utiliser par la suite.
Les premières lignes du code sont des informations pertinentes qui vont nous permettre d'implémenter la bibliothèque de NAO et *time* qui va mettre des intervalles pour que les lignes suivantes du code ne s'exécutent pas(6-7e ligne).

```
1   # -*- coding: utf-8 -*-
2   #127.0.0.1 adresse IP interne (standard)
3   #http://doc.aldebaran.com/1-14/naoqi/motion/control-walk.html
4   #http://doc.aldebaran.com/1-14/family/robots/joints_robot.html#robot-joints
5   #http://doc.aldebaran.com/1-14/naoqi/sensors/alsonar.html#alsonar
6   from naoqi import ALProxy
7   from time import sleep
8
9   ip = "127.0.0.1"
10  port = 9559
11
12  # ---- Proxy ---- #
13  motion = ALProxy("ALMotion",ip,port)
14  posture = ALProxy("ALRobotPosture", ip, port)
15  navigation = ALProxy("ALNavigation",ip,port)
16  module = ALProxy("ALSonar",ip,port)
17  memoire = ALProxy("ALMemory",ip,port)
```

Dans la partie du code suivant on va retrouver des fonctions simples, la première(23e ligne) va permettre d'activer une sécurité pour éviter que le robot puisse faire une collision.
La deuxième met NAO dans une position par défaut qui va lui permettre de par la suite bouger.
La troisième est celle qui va permettre au robot de se déplacer, on l'utilisera beaucoup par la suite.
Les prochaines sont moins intéressantes et m'ont permis de découvrir l'API.

```
19  # ---- Instruction ---- #
20  module.subscribe("myApplication") #Activer les sonars
21
22  """sonar et détection"""
23  def collision(collisionActive):
24      motion.setCollisionProtectionEnabled("Arms",collisionActive) #Eviter les collisions
25
26  def default():
27      """met le robot dans une position standard"""
28      posture.goToPosture("StandInit",0.5)
29
30  def marcher(axeX,axeY,axeZ,distSecur):
31      """Fonction pour marcher
32      axeX,Y en m
33      axeZ entre [-3.1415 to 3.1415]
34      Entree: 3x flt """
35      posture.goToPosture("StandInit", distObst)
36      navigation.setSecurityDistance(distSecur)
37      moveto = navigation.moveTo(axeX, axeY, axeZ)
38      print(moveto)
39
40  def bougerTete(angleZ,angleY):
41      """Bouge la tete
42      angleZ entre [-2.0857 to 2.0857]
43      angleY entre [-0.6720 to 0.5149]
44      Entrée: 2x float"""
45      motion.setStiffnesses("Head", 1.0)
46      names  = ["HeadYaw", "HeadPitch"]
47      angles  = [angleZ, angleY]
48      fractionMaxSpeed  = 0.2
49      motion.setAngles(names, angles, fractionMaxSpeed)
```

```
50
51  def bougerBrasDroit(EHautBas,EAvantArr,CoudeTour,CoudeFlex,Poignet,Main):
52      """RShoulderPitch   Right shoulder joint front and back (Y)  -2.0857 to 2.0857
53      RShoulderRoll   Right shoulder joint right and left (Z)  -1.3265 to 0.3142
54      RElbowYaw   Right shoulder joint twist (X)  -2.0857 to 2.0857
55      RElbowRoll  Right elbow joint (Z)   0.0349 to 1.5446
56      RWristYaw   Right wrist joint (X)   -1.8238 to 1.8238
57      RHand   Right hand  Open and Close"""
58      motion.setStiffnesses("RArm",1.0)
59      names = ["RShoulderPitch",
60               "RShoulderRoll",
61               "RElbowYaw",
62               "RElbowRoll",
63               "RWristYaw",
64               "RHand"]
65      angles = [EHautBas,EAvantArr,CoudeTour,CoudeFlex,Poignet,Main]
66      fractionMaxSpeed  = 0.2
67      motion.setAngles(names,angles,fractionMaxSpeed)
68
69  def bougerBrasGauche(EHautBas,EAvantArr,CoudeTour,CoudeFlex,Poignet,Main):
70      """LShoulderPitch   Left shoulder joint front and back (Y)  -2.0857 to 2.0857
71      LShoulderRoll   Left shoulder joint right and left (Z) -0.3142 to 1.3265
72      LElbowYaw   Left shoulder joint twist (X) -2.0857 to 2.0857
73      LElbowRoll  Left elbow joint (Z) -1.5446 to -0.0349
74      LWristYaw   Left wrist joint (X) -1.8238 to 1.8238
75      LHand   Left hand   Open and Close"""
76      motion.setStiffnesses("LArm",1.0)
77      names = ["LShoulderPitch",
78               "LShoulderRoll",
79               "LElbowYaw",
80               "LElbowRoll",
81               "LWristYaw",
82               "LHand"]
83      angles = [EHautBas,EAvantArr,CoudeTour,CoudeFlex,Poignet,Main]
84      fractionMaxSpeed  = 0.2
85      motion.setAngles(names,angles,fractionMaxSpeed)
```

5/ Résolution utilisée

Pour avoir le résultat final on, a utilisé un algorithme d'évitement simple, donc on lui donne une distance à avancer et s'il encontre un obstacle, il le dévi par la gauche ou la droite en fonction de là où se positionne l'obstacle et ensuite il repart tout droit.
Les *print()* dans le code suivant font nous permettre de déboguer ce que le robot fait, cela va n'avoir aucun impact sur ce qu'il fait et ce sera imprimé dans le terminal où on lui lance ses programmes.
Donc, dans un premier temps, on met le robot dans une position par défaut, on active les collisions pour qu'il s'arrête quand il est trop proche d'un objet, on lui donne une distance d'objet qui va être utile par la suite. Donc on lui demande d'avancer une certaine distance(93e ligne) et ensuite, on entre dans une boucle "tant que le robot bouge", on va donc récupérer les données des sonars pour permettre par la suite de contrôler si la distance d'objet est supérieure à celle des sonars.
Si c'est le cas, on arrête le robot, on le met dans une position par défaut et on fait les vérifications s'il doit tourner à gauche ou à droite en fonction du senseur le plus proche.
Enfin, on le fait avancer tout droit.

```
default() #Position qui permet de se dédplacer
collision(True) #Evité de faire des collisions
distObst = 0.40
#On le fait marcher tout droit
marcher(2,0,0,distObst)
print("Marche")

while motion.moveIsActive(): #Tant qu'il bouge

    #Valeurs en m des senseurs
    gaucheS = memoire.getData("Device/SubDeviceList/US/Left/Sensor/Value")
    droiteS = memoire.getData("Device/SubDeviceList/US/Right/Sensor/Value")

    #Bools pour savoir si un obstacle est proche à gauche et à droite
    distanceG = gaucheS < distObst
    distanceD = droiteS < distObst
    print(distanceD,distanceG)

    if distanceG or distanceD: #Si les bools sont vrais on fait les vérifs
        motion.stopMove()
        default()
        print("Arreter")

        distObst = 0.30 #On réduit la distance obstacle pour qu'il puisse réavancer

        if distanceG: #Si obstacle gauche on le fait tourner à droite
            marcher(0, -0.5, 0, distObst)
            print("Gauche")

        elif distanceD: #Si obstacle droit on le fait tourner à gauche
            marcher(0, 0.5, 0, distObst)
            print("Droite")

        marcher(0.5, 0, 0, distObst)
        print("Marche")

```

6/ Résolution voulu

La résolution la plus pertinente aurait été d'aller d'un point A à un point B et s'il encontre un obstacle il l'esquive et de rendre l'algorithme d'évitement plus approfondi, c'est-à-dire de pouvoir contourner un obstacle de toute taille, car celui utilisé permet seulement d'esquiver un obstacle d'une taille moyenne.

7/ Problèmes rencontrés

Les problèmes rencontrés ont été le changement de sujet, ou on été censé utiliser le logiciel de la société, Choregraphe, pour faire de la reconnaissance d'objet et que le robot puisse attraper ce fameux objet, mais à cause de problème avec ce dernier ceci n'a pas pu être fait. On a aussi le fait que l'API est très large et qu'on a utilisé très peu de fonctionnalités contrairement à ce qu'il y avait.