Référence des commandes du mbrobot

Il y a deux modules différents pour contrôler le robot Maqueen.

* Le module mbrobot permet de contrôler les moteurs de manière très facile avec les commandes forward(), left(), right(), backward(), stop(), leftArc() et rightArc().
* Importation du module
* Pour importer le module mbrobot, il faut utiliser l'instruction
* from mbrobot import \*
* Le module mbrobotmot permet de contrôler les moteurs de manière plus fine, de manière indépendante. Il permet donc de régler explicitement les vitesses des moteurs et de les contrôler séparément.
* Importation du module
* Pour importer le module mbrobotmot, il faut utiliser l'instruction
* from mbrobotmot import \*
* De nombreuses commandes ou fonctions sont communes aux deux modules, telles que les commandes setLED(), getDistance et les références aux capteurs infrarouges (irLeft et irRight) ainsi que les LEDs (ledLeft et ledRight)

Attention

Il faudrait éviter de charger les deux modules mbrobot et mbrobotmot dans un même programme. Le micro:bit n'a qu'une quantité très limitée de mémoire RAM, dans laquelle doit tenir l'interpréteur MicroPython et le programme a exécuter. L'importation des deux modules donne généralement lieu à des erreurs en raison de mémoire insuffisante.

# Commandes propres au module mbrobot

Le module mbrobot est le module qui permet de piloter le robot Maqueen le plus intuitivement. Il ne donne cependant pas accès au pilotage fin des moteurs.

Commandes du module mbrobot

|  |  |
| --- | --- |
| Syntaxe | Signification |
| forward() | fait avancer le robot en ligne droite |
| backward() | fait reculer le robot en ligne droite |
| left() | fait tourner le robot sur place (sur lui-même) vers la gauche |
| right() | fait tourner le robot sur place (sur lui-même) vers la droite |
| leftArc(radius) | déplace le robot sur un arc de cercle de rayon radius, vers la gauche |
| rightArc(radius) | déplace le robot sur un arc de cercle de rayon radius, vers la droite |
| stop() | arrête le robot |
| setSpeed(speed) | change la vitesse du robot à speed (vitesse par défaut : 50). Le paramètre speed est un nombre entier entre 0 et 255 qui indique la puissance électrique avec laquelle alimenter les moteurs. Les commanded forward(), backward(), right(), left(), rightArc() et leftArc() utilisent ensuite cette vitesse. |

0FDC9527-F0C0-4FBC-B94F-6C5B873B9318

# mbrobot.py  
# Version 2.3, Mar 4, 2020  
  
import gc  
from microbit import i2c, pin1, pin2, pin8, pin12, pin13, pin14, sleep  
import machine  
  
\_axe = 0.08  
  
def w(d1, d2, s1, s2):  
 try:  
 i2c.write(0x10, bytearray([0, d1, s1]))  
 i2c.write(0x10, bytearray([2, d2, s2]))  
 except:  
 print("Please switch on mbRobot!")  
 while True:  
 pass  
  
def setSpeed(speed):  
 global \_v  
 \_v = speed  
  
def forward():  
 w(0, 0, \_v, \_v)  
  
def backward():  
 w(1, 1, \_v, \_v)  
  
def stop():  
 w(0, 0, 0, 0)  
  
def right():  
 v = \_v   
 w(0 if \_v > 0 else 1, 1 if \_v > 0 else 0, v, v)   
  
def left():  
 v = \_v   
 w(1 if \_v > 0 else 0, 0 if \_v > 0 else 1, v, v)  
  
def rightArc(r):  
 v = abs(\_v) + 10  
 if r < \_axe:  
 v1 = 0  
 else:   
 f = (r - \_axe) / (r + \_axe) \* (1 - v \* v / 200000)   
 v1 = int(f \* v)  
 if \_v > 0:  
 w(0, 0, v, v1)  
 else:  
 w(1, 1, v1, v)  
  
def leftArc(r):  
 v = abs(\_v) + 10  
 if r < \_axe:  
 v1 = 0  
 else:  
 f = (r - \_axe) / (r + \_axe) \* (1 - v \* v / 200000)   
 v1 = int(f \* v)  
 if \_v > 0:  
 w(0, 0, v1, v)  
 else:  
 w(1, 1, v, v1)  
  
exit = stop  
delay = sleep  
  
def getDistance():  
 pin1.write\_digital(1)  
 pin1.write\_digital(0)  
 p = machine.time\_pulse\_us(pin2, 1, 50000)  
 cm = int(p / 58.2 + 0.5)  
 return cm if cm > 0 else 255  
  
def setLED(on):  
 pin8.write\_digital(on)  
 pin12.write\_digital(on)  
  
pin2.set\_pull(pin2.NO\_PULL)  
\_v = 50  
irLeft = pin13  
irRight = pin14  
ledLeft = pin8  
ledRight = pin12

# Commandes propres au module mbrobotmot

Le module mbrobotmot permet de gérer les moteurs du robot de manière plus fine. Il ne donne cependant la possibilité de contrôler des mouvements de haut niveau comme les arcs de cercles. En résumé, ce module permet de programmer tous les mouvements que l'on veut, mais il n'y a aucun mouvement pré-programmer. Il faut tous les programmer soi-même (pas de forward(), right(), leftArc() etc.)

Commandes du module mbrobotmot

|  |  |
| --- | --- |
| Syntaxe | Signification |
| motL.rotate(speed) | Fait tourner le moteur gauche à la vitesse indiquée par speed. Le paramètre speed est un nombre entier compris entre -255 et 255. Le moteur tourne dans le sens antihoraire si speed > 0 (contribue à faire avancer le robot), et dans le sens horaire si speed < 0 (contribue à faire reculer le robot). Le moteur est arrêté si speed = 0.  Attention  Pour utiliser motL, il faut importer le module mbrobotmot avec l'instruction  from mbrobotmot import \* |
| motR.rotate(speed) | Fait tourner le moteur droit à la vitesse indiquée par speed. Le paramètre speed est un nombre entier compris entre -255 et 255. Le moteur tourne dans le sens horaire si speed > 0 (contribue à faire avancer le robot), et dans le sens antihoraire si speed < 0 ((contribue à faire reculer le robot). Le moteur est arrêté si speed = 0.  Attention  Pour utiliser motR, il faut importer le module mbrobotmot avec l'instruction  from mbrobotmot import \* |

7F7FB28C-D1A8-438C-8C03-CA586D89D538

# mbrobotmot.py  
# Version 1.2, Aug 9, 2019  
  
import gc  
from microbit import i2c, pin1, pin2, pin8, pin12, pin13, pin14, sleep  
import machine  
  
class Motor:  
 def \_\_init\_\_(self, id):  
 self.\_id = 2 \* id  
  
 def rotate(self, s):  
 v = abs(s)  
 if s > 0:  
 self.\_w(0, v)   
 elif s < 0:  
 self.\_w(1, v)   
 else:   
 self.\_w(0, 0)   
  
  
 def \_w(self, d, s):  
 try:  
 i2c.write(0x10, bytearray([self.\_id, d, s]))  
 except:  
 print("Please switch on mbRobot!")  
 while True:  
 pass  
  
delay = sleep  
  
def getDistance():  
 pin1.write\_digital(1)  
 pin1.write\_digital(0)  
 p = machine.time\_pulse\_us(pin2, 1, 50000)  
 cm = int(p / 58.2 + 0.5)  
 return cm if cm > 0 else 255  
  
def setLED(on):  
 pin8.write\_digital(on)  
 pin12.write\_digital(on)  
  
pin2.set\_pull(pin2.NO\_PULL)  
irLeft = pin13  
irRight = pin14  
ledLeft = pin8  
ledRight = pin12  
motL = Motor(0)  
motR = Motor(1)

# Commandes communes aux modules mbrobot et mbrobotmot

Les commandes du tableau ci-dessous sont définies à la fois dans le module mbrobot et dans le module mbrobotmot. Elles permettent de contrôler tout ce qui n'est pas lié aux moteurs, notamment la lecture du capteur ultrasonique avec la fonction getDistance() ou des capteurs infrarouges disposés sous le châssis du robot (irLeft et irRight). Il permet également de contrôler les deux LEDs rouges à l'avant du robot ou de stoper un mouvement.

Commandes du module mbrobotmot

|  |  |
| --- | --- |
| Syntaxe | Signification |
| setLED(1) | allume les deux LEDs. |
| setLED(0) | éteint les deux LEDs |
| ledLeft.write\_digital(value) | Allume la LED de gauche si value vaut 1 et éteint la LED gauche si value vaut 0. |
| ledRight.write\_digital(value) | Allume la LED de droite si value vaut 1 et éteint la LED droite si value vaut 0. |
| irLeft.read\_digital() | retourne l’intensité lumineuse mesurée par le capteur infrarouge gauche du arobot |
| irRight.read\_digital() | retourne l’intensité lumineuse mesurée par le capteur infrarouge gauche du robot. |
| delay(ms) ou sleep(ms) | Met l’exécution du programme en pause et attend ms millisecondes avant de poursuivre son exécution avec la prochaine commande. |

# Mode simulé dans TigerJython

Le mode simulé n'est disponible que dans TigerJython. Il est possible d'influencer le mode simulé à l'aide de l'objet RobotContext qui est automatiquement chargé dans TigerJython avec le module mbrobot.

Les fonctionnalités du mode simulé peuvent se regrouper comme suit:

* Configurer le monde virtuel (sol virtuel, objets détectables par le capteur ultra-sonique, ...).
* Informations visuelles (trace, centre de rotation, cône de détection du capteur ultra-sonique, ...)
* Changer la position ou l'orientation du robot (comme si on le transportait à la main dans le monde réel)

## Méthode de RobotContext permettant de positionner le robot

Méthode de RobotContext permettant de positionner le robot

|  |  |
| --- | --- |
| Syntaxe | Signification |
| RobotContext.setStartPosition(  x: int,  y: int ) | Place le robot au point de coordonnées  Système de coordonnées  Il faut noter que le système d'axe n'est pas standard. L'origine se trouve au coin supérieur gauche du monde simulé et l'axe est orienté vers le bas. |
| RobotContext.setStartDirection(  direction: float ) | Règle l'orientation initiale du robot. Le paramètre direction représente un angle par rapport à l'axe .   * Pour direction=0, le robot regarde vers l'Est (vers la droite). * Pour direction=90, il regarde vers le Sud (vers le bas). * Pour direction=180, il regarde vers l'Ouest (vers la gauche) * Pour direction=270, il regarde vers le Nord (vers le haut) |

## Réglage des informations visuelles supplémentaires

Réglage des informations visuelles supplémentaires

|  |  |
| --- | --- |
| Syntaxe | Signification |
| RobotContext.enableTrace(  yes\_or\_no: bool ) | Active (yes\_or\_no=True) ou désactive (yes\_or\_no=False) la trace indiquant le trajet emprunté par le robot. |
| RobotContext.enableRotCenter(  yes\_or\_no: bool ) | Active (yes\_or\_no=True) ou désactive (yes\_or\_no=False) l'affichage du centre de rotation utilisé pour les états leftArc ou rightArc. |

## Configuration du monde virtuel

Configuration du monde virtuel

|  |  |
| --- | --- |
| Syntaxe | Signification |
| RobotContext.useBackground(  path: str ) | Charge le fichier image indiqué par le chemin relatif ou absolu path en tant que "sol virtuel". Les pixels colorés de ce "sol virtuel" influencent les valeurs lues par les capteurs IR virtuels, à savoir les valeurs de retour des fonctions irLeft.read\_digital() et irRight.read\_digital().  Sols virtuels  TigerJython met à disposition des sols virtuels dans le dossier sprites de la distribution TigerJython (archive JAR tigerjython2.jar).  Exemple de chargement d'images:  RobotContext.useBackground("sprites/blackarea.gif") |
| RobotContext.useTarget(  path: str,   mesh: list[tuple[int, int]],   x: int, y: int  ) | Permet d'indiquer le chemin vers un fichier image représentant un objet à détecter avec le capteur ultrason. Les paramètres x et y représentent les coordonnées auxquelles placer le centre de l'objet charger. Le paramètre mesh permet d'indiquer les coordonnées des sommets d'un polygone centré à l'origine qui réfléchira les ultrasons virtuels du capteur ultrasonique virtuel.  Exemple  Définition du maillage permettant au mode simulé de simuler la présence d'un objet  Définition du maillage permettant au mode simulé de simuler la présence d'un objet  mesh = [  (50,0),(25,43),(-25,43),  (-50,0),(-25,-43),(25,-43) ] RobotContext.useTarget("sprites/redtarget.gif",   mesh, 400, 400) |