Bibliothèque des classes de l'extension robotique

Ecole Robots
Apprendre à coder avec les robots éducatifs

ARTEC

Sommaire

1. Démarrer

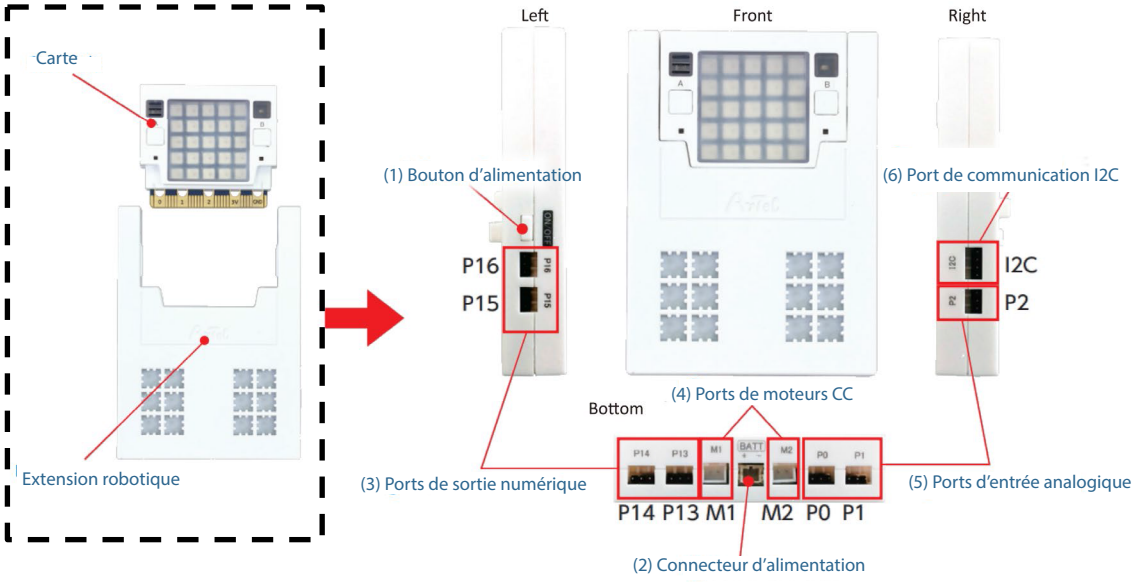
Ce manuel référence les classes de la bibliothèque d'ArtecRobo 2.0. Ce manuel requiert la maîtrise de quelques bases en langage Python.

2. ArtecRobo2.0

ArtecRobo2.0 désigne la carte ESP32 (Studuino:bit) et l'extension robotique.
MicroPython est l'un des environnements de programmation compatible avec Studuino:bit.

2.1. Configuration de l'extension robotique

Voici la configuration de l'extension robotique :



L'extension robotique fournit des ports additionnels que vous pouvez utiliser pour brancher des capteurs et moteurs à votre robots. Elle comporte des ports de sortie numériques, des ports pour moteurs CC, des ports d'entrée analogique et un port de communication I2C. Consultez le tableau ci-dessous pour voir quelles parties sont utilisables avec quels ports.

Port	Parts
Ports de sortie numérique	LED, Buzzers, servomoteurs
Ports de moteurs CC	Moteurs CC
Ports d'entrée analogique	Capteurs de lumière, capteurs de son, capteurs tactiles, photoréflecteurs IR
Port de communication I2C	Accéléromètres

2.2. MicroPython

MicroPython est une version de Python développée spécifiquement pour l'utiliser avec des microcontrôleurs. La syntaxe de programmation est identique à celle du Python 3.0, mais certaines bibliothèques Python ne sont pas utilisables en MicroPython parce qu'elles ont été conçues pour fonctionner avec la mémoire et le CPU limités d'un microcontrôleur. Cependant, certaines bibliothèques MicroPython spécifiques (comme la bibliothèque pour les microcontrôleurs GPIO) sont incluses comme un standard.
MicroPython a été largement étendu à d'autres microcontrôleurs et la bibliothèque Studuino:bit utilise une version spécifique de Studuino:bit.

3. Environnement de développement

uPyCraft est un des environnements de programmation utilisés par MicroPython. Voir la partie **6.2** du manuel "**Bibliothèque des classes de la carte ESP32**" pour savoir comment utiliser uPyCraft.

4. Bibliothèque des classes ArtecRobo 2.0

Voici la bibliothèque des classes d'ArtecRobo 2.0 :

Paquet	Module	Classe	Partie
pyatcrobo2	parts	DCMotor	Moteurs CC
		Servomotor	Servomoteurs
		Buzzer	Buzzers
		LED	LED
		LightSensor	Capteurs de lumière
		SoundSensor	Capteurs de son
		TouchSensor	Capteurs tactiles
		Temperature	Capteurs de température
		Accelerometer	Accéléromètres

4.1. La classe DCMotor

Cette classe sert à contrôler des moteurs CC.

4.1.1. Les constructeurs

Utilisez ce code pour créer des objets qui contrôlent des moteurs CC branchés sur un port des moteurs CC (M1 ou M2) de l'extension robotique.

```
from pyatcrobo2.parts import DCMotor
m1 = DCMotor('M1')
```

4.1.2. Contrôler un moteur CC

Utilisez la méthode **cw()** pour faire tourner un moteur CC dans le sens des aiguilles d'une montre et la méthode **ccw()** pour le faire tourner dans le sens contraire. La méthode **stop()** arrêtera le moteur, tandis que la méthode **brake()** permet d'arrêter complètement le mouvement du moteur. La méthode **power()** ajuste la vitesse du moteur. Réglez le paramètre **power** avec un nombre entre **0** et **255** pour choisir une vitesse.

```
from pyatcrobo2.parts import DCMotor
import time
m1 = DCMotor('M1')
m2 = DCMotor('M2')
m1.power(100)
m2.power(100)
m1.cw()
m2.cw()
time.sleep_ms(1000)
m1.stop()
m2.stop()
```

Ce code fera tourner les moteurs connectés aux ports M1 et M2 dans le sens des aiguilles d'une montre à la vitesse de 100 pendant 1 seconde avant de les arrêter.

4.2. La classe Servomotor

Cette classe sert à contrôler les servomoteurs.

4.2.1. Les constructeurs

Utilisez ce code pour créer des objets qui contrôlent des servomoteurs branchés sur un port spécifique (P13, P14, P15 ou P16) sur l'extension robotique.

```
from pyatcrobo2.parts import Servomotor
sv13 = Servomotor('P13')
```

4.2.2. Contrôler des servomoteurs

Utilisez la méthode **set_angle(degree)** pour régler l'angle d'un servomoteur.

```
from pyatcrobo2.parts import Servomotor
import time
sv13 = Servomotor('P13')
sv13.set_angle(0)
time.sleep_ms(1000)
sv13.set_angle(180)
```

Ce code fera tourner le servomoteur connecté au port P13 à 0°, puis à 180° une seconde plus tard.

4.2.3. Libérer des PWM

Le Studuino:bit utilise PWM pour régler les angles du servomoteur. Il peut utiliser jusqu'à 4 PWM simultanément. Si vous utilisez 5 éléments ou plus qui utilisent des PWM, vous ne pourrez utiliser aucun des objets Servomotor créés. Ce problème peut être résolu avec la méthode **release()** pour libérer un PMW assigné à un autre objet Servomotor.

4.3. La classe Buzzer

Cette classe sert à contrôler les buzzers.

4.3.1. Les constructeurs

Utilisez ce code pour créer des objets qui contrôlent des buzzers branchés sur un port spécifique (P13, P14, P15 ou P16) sur l'extension robotique.

```
from pyatcrobo2.parts import Buzzer
bz13 = Buzzer('P13')
```

4.3.2. Contrôler des buzzers

Utilisez la méthode **on(sound, *, volume=None, duration=None)** pour lancer un buzzer. Utilisez le paramètre **sound** pour régler la fréquence du son avec le nombre d'une note MIDI ou le nom d'une note. Pour une liste des nombres et des noms de note utilisables, voir la partie **5.1**. Le paramètre **volume** vous permet de régler le volume entre 0 et 100. Le paramètre **duration** est utilisé pour régler la longueur du son de sortie en millisecondes. Si le paramètre **duration** reste non spécifié, la méthode **off()** ne pourra pas être utilisée pour arrêter le buzzer.

```
from pyatcrobo2.parts import Buzzer
bz13 = Buzzer('P13')
bz13.on('C4', duration=1000)
```

Ce code fait jouer au buzzer branché sur le port P13 de l'extension robotique la note C4 pendant 1 seconde.

4.3.3. Libérer des PWM

Le Studuino:bit utilise PWM pour ses sons de sortie. Il peut utiliser jusqu'à 4 PWM simultanément. Si vous utilisez 5 éléments ou plus qui utilisent des PWM, vous ne pourrez utiliser aucun des objets Buzzer créés. Ce problème peut être résolu avec la méthode **release()** pour libérer un PMW assigné à un autre objet Buzzer.

4.4. La classe des LED

Cette classe sert à contrôler les LED.

4.4.1. Les constructeurs

Utilisez ce code pour créer des objets qui contrôlent des LED branchées à un port spécifique (P13, P14, P15 ou P16) sur l'extension robotique.

```
from pyatcrobo2.parts import LED
led13 = LED( 'P13' )
```

4.4.2. Contrôler des LED

Utilisez la méthode **on()** pour allumer une LED et la méthode **off()** pour l'éteindre.

```
from pyatcrobo2.parts import LED
import time
led13 = LED( 'P13' )
while True:
    led13.on()
    time.sleep_ms(500)
    led13.off()
    time.sleep_ms(500)
```

Ce code allumera et éteindra la LED connectée au port P13 de l'extension robotique toutes les 500 millisecondes.

4.5. La classe IRPhotoreflector

Cette classe sert à contrôler des photoréflecteurs IR.

4.5.1. Les constructeurs

Utilisez ce code pour créer des objets qui contrôlent des photoréflecteurs IR branchés à un port spécifique (P0, P1, P2 ou P3) sur l'extension robotique.

```
from pyatcrobo2.parts import IRPhotoReflector
sensor0 = IRPhotoReflector( 'P0' )
```

4.5.2. Trouver les valeurs du photoréflecteur IR

Utilisez la méthode **get_value()** pour trouver la valeur analogique d'un photoréflecteur IR. Elle sera formatée en nombre entier entre 0 et 4095.

```
from pyatcrobo2.parts import IRPhotoReflector
import time
sensor0 = IRPhotoReflector( 'P0' )
while True:
    print(sensor0.get_value())
    time.sleep_ms(500)
```

Ce code affichera la valeur du photoréflecteur IR branché sur le port P0 de l'extension robotique sur le terminal toutes les 500 millisecondes.

4.6. La classe LightSensor

Cette classe sert à contrôler des capteurs de lumière.

4.6.1. Les constructeurs

Utilisez ce code pour créer des objets qui contrôlent des capteurs de lumière branchés sur un port spécifique (P0, P1, P2 ou P3) de l'extension robotique.

```
from pyatcrobo2.parts import LightSensor
sensor0 = LightSensor( 'P0' )
```

4.6.2. Trouver les valeurs du capteur de lumière

Utilisez la méthode **get_value()** pour trouver la valeur analogique d'un capteur de lumière. Il sera formaté en nombre entier entre 0 et 4095.

```
from pyatcrobo2.parts import LightSensor
import time
sensor0 = LightSensor( 'P0' )
while True:
    print(sensor0.get_value())
    time.sleep_ms(500)
```

Ce code affichera la valeur du capteur de lumière branché sur le port P0 de l'extension robotique sur le terminal toutes les 500 millisecondes.

4.7. La classe SoundSensor

Cette classe sert à contrôler des capteurs de son.

4.7.1. Les constructeurs

Utilisez ce code pour créer des objets qui contrôlent des capteurs de son branchés sur un port spécifique (P0, P1, P2 ou P3) de l'extension robotique.

```
from pyatcrobo2.parts import SoundSensor
sensor0 = SoundSensor( 'P0' )
```

4.7.2. Trouver les valeurs du capteur de son

Utilisez la méthode **get_value()** pour trouver la valeur analogique d'un capteur de son. Elle sera formatée en nombre entier entre 0 et 4095.

```
from pyatcrobo2.parts import SoundSensor
import time
sensor0 = SoundSensor( 'P0' )
while True:
    print(sensor0.get_value())
    time.sleep_ms(500)
```

Ce code affichera la valeur du capteur de son branché sur le port P0 de l'extension robotique sur le terminal toutes les 500 millisecondes.

4.8. La classe TouchSensor

Cette classe sert à contrôler des capteurs tactiles.

4.8.1. Les constructeurs

Utilisez ce code pour créer des objets qui contrôlent des capteurs tactiles branchés sur un port spécifique (P0, P1, P2 ou P3) de l'extension robotique.

```
from pyatcrobo2.parts import TouchSensor
sensor0 = TouchSensor( 'P0' )
```

4.8.2. Trouver les valeurs du capteur tactile

Utilisez la méthode **get_value()** pour trouver la valeur numérique d'un capteur tactile. La valeur sera **0** ou **1**. La méthode **is_pressed()** retournea **True/False** pour indiquer si le capteur tactile a été pressé ou non.

```
from pyatcrobo2.parts import TouchSensor
import time
sensor0 = TouchSensor( 'P0' )
while True:
    print(sensor0.get_value())
    print(sensor0.is_pressed())
    time.sleep_ms(500)
```

Ce code indiquera si le capteur tactile branché sur le port P0 de l'extension robotique est pressé ou non sur le terminal toutes les 500 millisecondes. Si le capteur tactile a été pressé, il affichera 0 et True, sinon 1 et False.

4.9. La classe Temperature

Cette classe sert à contrôler des capteurs de température.

4.9.1. Les constructeurs

Utilisez ce code pour créer des objets qui contrôlent des capteurs de température branchés sur un port spécifique (P0, P1, P2 ou P3) de l'extension robotique.

```
from pyatcrobo2.parts import Temperature
sensor0 = Temperature( 'P0' )
```

4.9.2. Trouver les valeurs du capteur de température

Utilisez la méthode **get_value()** pour trouver la valeur analogique d'un capteur de température. Elle sera formatée en nombre entier entre 0 et 4095. La méthode **get_celsius()** sert à trouver une valeur de capteur de température en degrés celsius.

```
from pyatcrobo2.parts import Temperature
import time
sensor0 = Temperature( 'P0' )
while True:
    print(sensor0.get_value())
    print(sensor0.get_celsius())
    time.sleep_ms(500)
```

Ce code affichera sur le terminal la valeur analogique et la température en celsius du capteur de température branché sur le port P0 toutes les 500 millisecondes.

Cette classe sert à contrôler des accéléromètres.

4.10. La classe Accelerometer

Cette classe sert à contrôler des accéléromètres.

4.10.1. Les constructeurs

Utilisez ce code pour créer des objets qui contrôlent des accéléromètres branchés sur le port I2C de l'extension robotique.

```
from pyatcrobo2.parts import Accelerometer
sensor0 = Accelerometer( 'I2C' )
```

L'accéléromètre a une pleine échelle (accélération maximale mesurable) de ± 2G, montrée en unité de G.

4.10.2. Trouver les valeurs de l'accéléromètre

Utilisez la méthode **get_value()** pour trouver les valeurs de votre accéléromètre. Les valeurs seront retournées en format tuple (x,y,z). Les méthodes **get_x(), get_y()**, **get_z()** servent à trouver l'accélération le long des axes x, y et z séparément.

```
from pyatcrobo2.parts import Accelerometer
import time
sensor0 = Accelerometer( 'I2C' )
while True:
    print(sensor0.get_values())
    time.sleep_ms(500)
```

Ce code affichera la valeur de l'accéléromètre branché sur le port I2C de l'extension robotique sur le terminal toutes les 500 millisecondes.

5. Annexes

5.1. Son de sortie

Voici les nombres et les noms des notes MIDI utilisés dans la classe **Buzzer** :

MIDI	Note	MIDI	Note	MIDI	Note	MIDI	Note	MIDI	Note	MIDI	Note	MIDI	Note
48	C3	60	C4	72	C5	84	C6	96	C7	108	C8	120	C9
49	CS3	61	CS4	73	CS5	85	CS6	97	CS7	109	CS8	121	CS9
50	D3	62	D4	74	D5	86	D6	98	D7	110	D8	122	D9
51	DS3	63	DS4	75	DS5	87	DS6	99	DS7	111	DS8	123	DS9
52	E3	64	E4	76	E5	88	E6	100	E7	112	E8	124	E9
53	F3	65	F4	77	F5	89	F6	101	F7	113	F8	125	F9
54	FS3	66	FS4	78	FS5	90	FS6	102	FS7	114	FS8	126	FS9
55	G3	67	G4	79	G5	91	G6	103	G7	115	G8	127	G9
56	GS3	68	GS4	80	GS5	92	GS6	104	GS7	116	GS8		
57	A3	69	A4	81	A5	93	A6	105	A7	117	A8		
58	AS3	70	AS4	82	AS5	94	AS6	106	AS7	118	AS8		
59	B3	71	B4	83	B5	95	B6	107	B7	119	B8		

5.2. Tableau des classes

Voici la bibliothèque complète d'ArtecRobo2 :

Fonction	Classe/Module	Méthode/Attribut	Instructions
Moteurs CC	DCMotor	_init_(pin)	Utilisez le paramètre **pin** pour spécifier la broche M1 ou M2 et créer une instance moteur CC. dcm=DCMotor('M1')
		cw()	Fait tourner un moteur CC dans le sens des aiguilles d'une montre.
		ccw()	Fait tourner un moteur CC dans le sens contraire des aiguilles d'une montre.
		stop()	Fait cesser de tourner un moteur CC .
		brake()	Arrête complètement la rotation d'un moteur CC .
		power(power)	Règle le paramètre **power** sur un nombre entre 0 et 255 pour changer la vitesse d'un moteur CC.
Servomoteurs	Servomotor	_init_(pin)	Utilise le paramètre **pin** pour spécifier la broche P13, P14, P15 ou P16 et créer une instance Servomotor. sv=ServoMotor('P13')
		set_angle(degree)	Règle le paramètre **degree** à un angle entre 0 et 180 pour changer l'angle du servomoteur. sv.set_angle(90)
		release()	Libère un PWM assigné à un objet Servomotor.
Buzzers	Buzzer	_init_(pin)	Utilise le paramètre **pin** pour spécifier la broche P13, P14, P15 ou P16 et créer une instance Buzzer. bzr=Buzzer('P13')

		on(sound, *,
		volume=-1,
		duration=-1)
			Règle le son du buzzer dans les paramètres **sound, duration** et **volume**. Le son peut être réglé avec des noms de note (C3-G9), des notes MIDI (48-127) ou une fréquence (comme un entier). Le volume peut être réglé entre 1 et 100 et la durée en ms. Si le paramètre **duration** reste non spécifién le buzzer continuera de jouer jusqu'à ce que la méthode **off()** soit appelée. Si le paramètre **volume** reste non spécifié, le volume sera ajusté à la tonalité du son.
			bzr.on('50', duration=1000) # Le buzzer jouera la note correspondant à la note MIDI 50 pendant 1 seconde.
			bzr.on('C4', volume=1, duration=1000) # Le buzzer jouera la note C4 au volume 1 pendant 1 seconde.
			bzr.on(440)

		off()	Arrête le son du buzzer. bzr.off()
		release()	Libère un PWM assigné à un objet **Buzzer**.
LED	LEDs	_init_(pin)	Utilise le paramètre **pin** pour spécifier la broche P13, P14, P15 ou P16 et créer une instance LED. led=LED('P13')
		on()	Allume une LED led.on()
		off()	Eteint une LED led.off()

Photoréflecteurs IR	IRPhotoReflector	_init_(pin) get_value()	Utilise le paramètre **pin** pour spécifier la broche P0, P1 ou P2 et créer une instance IR Photoreflector. irp=IRPhotoReflector('P0') Récupère la valeur du capteur (0-4095). irp.get_value()
Capteurs de lumière	LightSensor	_init_(pin)	Utilise le paramètre **pin** pour spécifier la broche P0, P1 ou P2 et créer une instance Light Sensor. ls=LightSensor('P0')
		get_value()	Récupère la valeur du capteur (0-4095). ls.get_value()
Capteur de température	Temperature	_init_(pin)	Utilise le paramètre **pin** pour spécifier la broche P0, P1 ou P2 et créer une instance Temeprature Sensor. ts=Temperature('P0')
		get_value()	Récupère la valeur du capteur (0-4095). ts.get_value()
		get_celsius()	Récupère la valeur du capteur de température en degrés celsius. ts.get_celsius()
Capteurs de son	SoundSensor	_init_(pin)	Utilise le paramètre **pin** pour spécifier la broche P0, P1 ou P2 et créer une instance Sound Sensor. ss=SoundSensor('P0')
		get_value()	Récupère la valeur du capteur (0-4095). ss.get_value()
Capteurs tactiles	TouchSensor	_init_(pin)	Utilise le paramètre **pin** pour spécifier la broche P0, P1 ou P2 et créer une instance Touch Sensor. ss=TouchSensor('P0')
		get_value()	Récupère la valeur du capteur (0 ou 1). ss.get_value()
		is_pressed()	Trouve l'état actuel du capteur. Retourne **True** si le capteur tactile est en train d'être pressé.
Accéléromètres	Accelerometer	_init_(pin)	Utilise le paramètre **pin** pour spécifier la broche I2C et créer une instance Accelerometer. acc=Accelerometer('I2C')
		get_values()	Trouve l'accélération le long des axes x, y et z. L'intervalle de mesure maximale est ±2G et les unités sont en G. acc.get_values()
		get_x	= get_values()[0]
		get_y	= get_values()[1]
		get_z	= get_values()[2]