Document V1.0 – Pyqueen V1.0

Développement du module Pyqueen

Utilisation du robot micro:Maqueen de DFRobot sous Python à l'aide d'un module Python adapté au cadre scolaire

THOMAS POCHART
03/01/2020

Sommaire

Introduction

Le robot micro:Maqueen de DFRobot est un robot à roues commercialisé depuis 2018, visant à introduire la robotique et la programmation à de jeunes étudiants en STEM (Science, Technologie, Engineering and Mathematics). Fier d'un succès international, le robot connaît quatre versions successives, toutes basées sur l'interaction avec le microordinateur BBC micro:bit.
A l'heure où ces lignes sont écrites (fin décembre 2019), le robot n'est compatible qu'avec MakeCode.

Le présent document détaille l'élaboration du module Pyqueen, qui permet à des étudiants de coder le robot avec Python dans un environnement aseptisé [4].

I. Prérequis pour la communication entre la carte BBC micro:bit et le robot Maqueen

1. Acquisition de l'adresse I2C du robot micro:Maqueen

La carte BBC micro:bit va communiquer avec le corps du robot Maqueen via le bus informatique i2c. Il est donc nécessaire de déterminer l'adresse i2c du corps du robot Maqueen afin de pouvoir lui envoyer des signaux avec la carte BBC micro:bit.

Dans ce but, on va utiliser la carte BBC micro:bit afin de scanner les périphériques i2c lui étant connectés et faire afficher sur son écran LED les adresses obtenues en hexadécimal. Le code-source employé est le suivant :

```
from microbit import *

while True:
    L=i2c.scan()
    for x in L:
        display.scroll(hex(x))
```

Exécuter ce code sans connecter la carte BBC micro:bit au robot Maqueen renvoie les adresses :

0x19
0x1E

Exécuter ce code en connectant la carte BBC micro:bit au robot Maqueen renvoie les adresses :

0x10
0x19
0x1E

On en déduit que l'adresse i2c du robot Maqueen est 0x10. Cette adresse sera définie comme une variable globale dans tout le module pyqueen, puisqu'elle sera utilisée pour chaque fonction :

```
from microbit import *
from machine import *

addr=0x10
```

Remarque : Les adresses 0x19 et 0x1E renvoient respectivement à l'accéléromètre et au magnétomètre montés sur la carte BBC micro:bit, qui s'avère ici être une variant 1 [1].

2. Pins du robot Maqueen

Le robot Maqueen présente plusieurs composants électroniques avec lesquels la carte BBC micro:bit est susceptible d'interagir. Chacun de ses composants est associé à un pin, dont les numéros (lisibles sur la carte électronique à l'œil nu) sont résumés dans le tableau ci-dessous :

Numéro du pin	Composant associé
1	TRIG (émetteur d'ultrasons)
2	ECHO (récepteur d'ultrasons)
8	LED-L (LED rouge avant gauche)
12	LED-R (LED rouge avant droite)
13	LINE-L (Capteur de suivi de ligne gauche)
14	LINE-R (Capteur de suivi de ligne droit)
15	RGB (LEDs à couleurs réglables sous le chassis)

Pins des différents composants présents sur la carte électronique constituant le corps du robot Maqueen

II. Utilisation des moteurs

1. Etude du script JavaScript fourni par DFRobot pour l'utilisation des moteurs du Maqueen dans MakeCode

Le module officiel Maqueen pour MakeCode, écrit en JavaScript, fournit la fonction suivante pour l'utilisation des moteurs du Maqueen :

```
    /**
     * Set the direction and speed of Maqueen motor.
     */

    //% weight=90
    //% blockId=motor_MotorRun block="motor|%index|move|%Dir|at
speed|%speed"
    //% speed.min=0 speed.max=255
    //% index.fieldEditor="gridpicker" index.fieldOptions.columns=2
    //% direction.fieldEditor="gridpicker" direction.fieldOptions.columns=2
    export function motorRun(index: Motors, direction: Dir, speed: number):
void {
        let buf = pins.createBuffer(3);
        if (index == 0) {
            buf[0] = 0x00;
            buf[1] = direction;
            buf[2] = speed;
            pins.i2cWriteBuffer(0x10, buf);
        }
        if (index == 1) {
            buf[0] = 0x02;
            buf[1] = direction;
            buf[2] = speed;
            pins.i2cWriteBuffer(0x10, buf);
        }
        if (index == 2) {
            buf[0] = 0x00;
            buf[1] = direction;
            buf[2] = speed;
            pins.i2cWriteBuffer(0x10, buf);
            buf[0] = 0x02;
            pins.i2cWriteBuffer(0x10, buf);
        }
    }
```

Les enums Motors et Dir ont été précédemment définis :

```
    export enum Motors {
        //% blockId="left motor" block="left"
        M1 = 0,
        //% blockId="right motor" block="right"
        M2 = 1,
        //% blockId="all motor" block="all"
        All = 2
    }
```

```
    export enum Dir {
        //% blockId="CW" block="Forward"
        CW = 0x0,
        //% blockId="CCW" block="Backward"
        CCW = 0x1
    }
```

De ce code, on peut déduire plusieurs informations :

· Commander les moteurs se fait en écrivant directement à l'adresse i2c un vecteur de taille 3.

· Le premier élément de ce vecteur est 0x00 ou 0x02. Selon la valeur du paramètre Motors pris en entrée par la fonction motorRun, le vecteur est écrit avec comme premier élément 0x00, avec comme premier élément 0x02, ou les deux successivement. L'enum Motors désignant quel moteur(s) on souhaite contrôler avec la fonction, il vient que le premier élément du vecteur désigne quel moteur est à mettre en mouvement. Le cas où le moteur à déplacer est le gauche correspond à la valeur 0 de l'index Motors, donc à un premier élément du vecteur à écrire valant : 0x00. De même, le premier élément 0x02 correspond au moteur droit.

· Le deuxième élément de ce vecteur est direction, qui est un Dir. La définition de l'enum Dir apprend que direction ne peut prendre que deux valeurs : 0x0 pour faire avancer le Maqueen, 0x1 pour le faire reculer.

· Le troisième élément de ce vecteur est speed, un nombre qui doit d'après les commentaires précédant la définition de la fonction motorRun être compris entre 0 (vitesse minimale) et 255 (vitesse maximale).

2. Utilisation du moteur gauche

Nous allons écrire une fonction Python motorL reproduisant l'appel de la fonction JavaScript motorRun afin de faire tourner le moteur de gauche.

Dans le but de didactiser le plus possible la fonction motorL, elle ne prendra qu'un seul argument : une consigne entière comprise entre -100 et 100. La valeur absolue de cette consigne indiquera la vitesse de rotation du moteur (sur une échelle de 0 à 100 qui sera plus familière aux élèves qu'une échelle de 0 à 255), tandis que le signe de cette consigne indiquera le sens de rotation du moteur.

En langage naturel pseudo-Python, la fonction motorL est donc :

```
Définir fonction motorL(n) :
        Si n>=0 :
                Direction prend la valeur 0x0
        Sinon :
                Direction prend la valeur 0x1
        Ecrire par i2c à l'adresse 0x10 le vecteur [0x0,0x1,valeur absolue de n*100//255]
```

Puisque cette fonction est destinée à être utilisée par des élèves, on ajoutera quelques garde-fous :

· Appeler la fonction avec une consigne supérieure à 100 aura pour effet l'appel de la fonction avec une consigne valant 100.
· Appeler la fonction avec une consigne inférieure à -100 aura pour effet l'appel de la fonction avec une consigne valant -100.
· Appeler la fonction avec une consigne non entière aura pour effet l'appel de la fonction avec la partie entière de cette consigne.

L'implémentation Python de motorL est alors :

```
def motorL(n):
        if n>100:
                n=100
        if n<-100:
                n=-100
        if n>=0:
                direc=0x0
        else:
                direc=0x1
        v=abs(floor(n))*255//100
        i2c.write(addr,bytearray([0x0,direc,v]))
```

3. Utilisation du moteur droit

On implémente une fonction motorR analogue à motorL, mais agissant sur le moteur droit :

```
def motorR(n):
      if n>100:
            n=100
      if n<-100:
            n=-100
      if n>=0:
            direc=0x0
      else:
            direc=0x1
      v=abs(floor(n))*255//100
      i2c.write(addr,bytearray([0x2,direc,v]))
```

4. Utilisation simultanée des deux moteurs

L'appel simultané des fonctions motorL et motorR à même consigne met en mouvement le robot, mais celui-ci ne suit pas une ligne droite : il dérive vers la droite.
Une fonction motors prenant en argument une consigne n du même type que celle donnée à motorL et motorR ne devra donc pas appeler motorL(n) et motorR(n), mais motorL(f(n)) et motorR(n) avec :

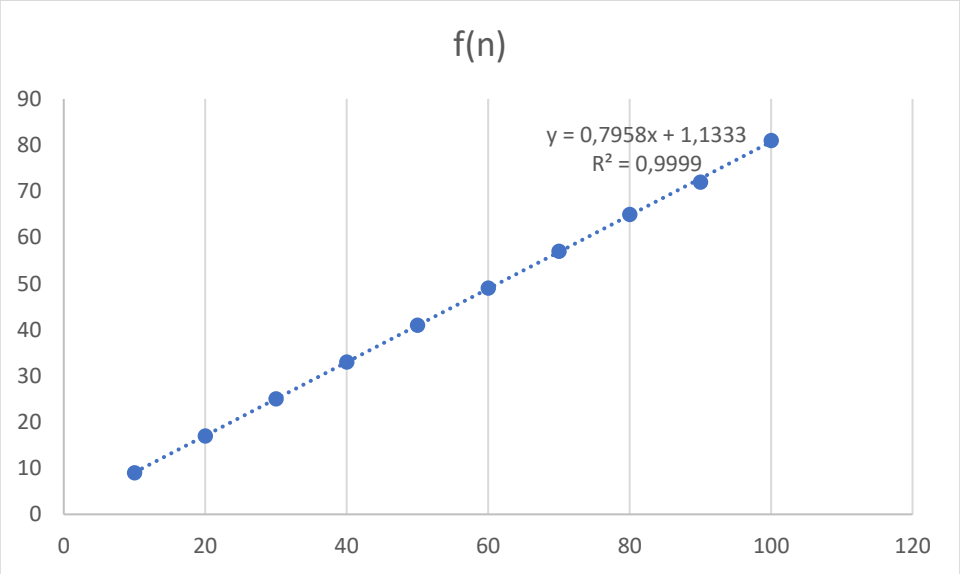
$$|f(n)| \leq |n|$$

Expérimentalement, on trouve :

n	f(n)
10	9
20	17
30	25
40	33
50	41
60	49
70	57
80	65
90	72
100	81

Valeurs mesurées de $f(n)$ pour des valeurs de n multiples de 10.

Excel permet de trouver une approximation de f par une fonction affine :



On retiendra comme définition pour f :

$$\begin{cases} f(0) = 0 \\ \forall n \in [\![1;100]\!], f(n) = \left\lfloor \frac{4}{5}n \right\rfloor + 1 \\ \forall n \in [\![-100;-1]\!], f(n) = -f(-n) \end{cases}$$

D’où l’implémentation Python de la fonction motors :

```
def motors(n):
      if n==0:
            f=0
      elif n>0:
            f=floor(4*n/5)+1
      else:
            f=-floor(-4*n/5)-1
      motorL(f)
      motorR(n)
```

III. Utilisation de l'émetteur/récepteur d'ultrasons afin d'acquérir la distance de l'objet frontal le plus proche

1. Etude du script JavaScript fourni par DFRobot pour l'acquisition de la distance de l'objet frontal le plus proche via l'émetteur/récepteur d'ultrasons

Le module officiel Maqueen pour MakeCode, écrit en JavaScript, fournit la fonction suivante pour l'utilisation du système ultrasonique du Maqueen :

```
    /**
    * Read ultrasonic sensor.
    */

    //% blockId=ultrasonic_sensor block="read ultrasonic sensor |%unit "
    //% weight=95
    export function Ultrasonic(unit: PingUnit, maxCmDistance = 500): number
{
        let d
        pins.digitalWritePin(DigitalPin.P1, 0);
        if (pins.digitalReadPin(DigitalPin.P2) == 0) {
            pins.digitalWritePin(DigitalPin.P1, 1);
            pins.digitalWritePin(DigitalPin.P1, 0);
            d = pins.pulseIn(DigitalPin.P2, PulseValue.High, maxCmDistance
* 58);
        } else {
            pins.digitalWritePin(DigitalPin.P1, 0);
            pins.digitalWritePin(DigitalPin.P1, 1);
            d = pins.pulseIn(DigitalPin.P2, PulseValue.Low, maxCmDistance *
58);
        }
        let x = d / 39;
        if (x <= 0 || x > 500) {
            return 0;
        }
        switch (unit) {
            case PingUnit.Centimeters: return Math.round(x);
            default: return Math.idiv(d, 2.54);
        }

    }
```

De ce code, on déduit plusieurs informations :

· L'envoi d'un son via l'émetteur à ultrasons, situé sur le pin 1, se fait en écrivant via i2c sur ce pin les nombres 0, 1, 0 dans cet ordre. Il apparaît que lorsque le nombre écrit sur le pin 1 est 0, l'émetteur n'émet pas, et lorsque le nombre écrit sur le pin 1 est 1, l'émetteur émet.
· Le pin 2 renvoie un digital pouvant prendre deux valeurs correspondant à ses deux états : High (un ultrason est capté) et Low (il n'y a pas d'ultrason capté).

Il convient d'expliquer comment la distance à l'objet frontal le plus proche est récupérée de la durée de l'impulsion mesurée.
Notons d cette distance.

Supposons que l'émetteur émette une impulsion sonore dans un milieu silencieux. Le temps mis par l'impulsion pour quitter l'émetteur, rebondir sur l'objet frontal le plus proche, et revenir au récepteur est, en notant c_s la célérité du son dans l'air :

$$T = 2d/c_s$$

Donc, en faisant l'acquisition de T, on peut obtenir d via la formule :

$$d = Tc_s/2$$

Remarquons que la boucle conditionnelle présente sur la fonction Ultrasonic permet de mesurer la durée d'une impulsion d'ultrasons si l'émetteur est initialement silencieux, et d'une impulsion de silence si l'émetteur est initialement en train d'émettre un ultrason.
Raisonnons également sur ce deuxième cas.
On suppose qu'à $t = 0$, l'émetteur cesse d'émettre. Le capteur détecte donc du silence. L'émetteur recommence à émettre peu après, à un instant t_1. Le son qui est émis à cet instant par l'émetteur arrive au récepteur après une durée :

$$T = 2d/c_s$$

C'est-à-dire à l'instant $t_1 + T$.
Supposons que l'on ait $t_1 \ll T$, hypothèse correspondant au fait que l'impulsion est de durée courte devant la durée de l'expérience (d'où son nom d'impulsion). Dans ce cas, $t_1 + T \approx T$.
Alors, la durée du silence mesuré par le récepteur est assimilable à T, et on en déduit d comme précédemment.

2. Implémentation Python de la fonction distance

On va coder une fonction distance ne prenant pas d'argument et renvoyant la distance en cm de l'objet frontal le plus proche. L'approche choisie sera une version simplifiée de la fonction faite par DFRobot, puisque dans le cas où l'émetteur est initialement en fonctionnement, la fonction l'éteindra et attendra que le silence soit détecté par le récepteur.

Estimons le temps d'attente nécessaire.
Dans le cas d'un objet à $d = 1\ m$, en prenant $c_s = 340\ m/s$, $T \approx 5\ ms$.
Donc, attendre 10 ms devrait suffire à ce que le récepteur détecte que l'émetteur n'émet plus (on fait l'hypothèse que le milieu ne présente pas d'autres ultrasons détectables par le récepteur, condition de toute façon nécessaire à l'utilisation de la fonction distance).
On déduit également de ce calcul que si le récepteur ne reçoit rien après 10ms d'attente, on peut arrêter l'écoute et renvoyer une distance fictive.

En langage naturel pseudo-Python, cette fonction s'écrit :

Définir fonction distance() :
    Si l'état du pin 1 est High :
        Faire passer l'état du pin 1 sur Low
        Attendre 10 ms
    Faire passer l'état du pin 1 sur High
    Faire passer l'état du pin 1 sur Low
    Mesurer la durée T en µs pendant laquelle l'état du pin 2 est High : si cette mesure n'est pas achevée après 10ms, affecter -1 à T.
    Si T<0 :
        Renvoyer(1000000) #Distance fictive d'un million de cm
    Sinon :
        Renvoyer 340T/20000

Son implémentation Python est :

```
def distance():
    if pin1.read_digital()==1:
        pin1.write_digital(0)
        sleep(10)
    pin1.write_digital(1)
    pin1.write_digital(0)
    pin2.read_digital()
    T=time_pulse_us(pin2, 1, 10000)
    if T<0:
        return(1000000)
    else:
        d=340*T/20000
        return(d)
```

IV. Utilisation des LEDs avant

1. Etude du script JavaScript fourni par DFRobot pour l'utilisation des LEDs avant

Le module officiel Maqueen pour MakeCode, écrit en JavaScript, fournit la fonction suivante pour l'utilisation des LEDs avant du Maqueen :

```
    /**
     * Turn on/off the LEDs.
     */

    //% weight=20
    //% blockId=writeLED block="LEDlight |%led turn |%ledswitch"
    //% led.fieldEditor="gridpicker" led.fieldOptions.columns=2
    //% ledswitch.fieldEditor="gridpicker" ledswitch.fieldOptions.columns=2
    export function writeLED(led: LED,ledswitch: LEDswitch): void {
        if (led == LED.LEDLeft) {
            pins.digitalWritePin(DigitalPin.P8, ledswitch)
        } else if (led == LED.LEDRight) {
            pins.digitalWritePin(DigitalPin.P12, ledswitch)
        } else {
            return
        }
    }
```

Les Enums LED et LEDswitch ont été définis par :

```
    export enum LED {
        //% blockId="LEDLeft" block="left"
        LEDLeft = 8,
        //% blockId="LEDRight" block="right"
        LEDRight = 12
    }
```

```
    export enum LEDswitch {
        //% blockId="turnOn" block="ON"
        turnOn = 0x01,
        //% blockId="turnOff" block="OFF"
        turnOff = 0x00
    }
```

De ce code, on peut tirer plusieurs informations :

· On retrouve le fait que le pin de la LED de gauche est 8 et celui de la LED de droite est 12.

· Pour allumer une LED, il faut écrire sur le pin correspondant 0x01. Pour éteindre une LED, il faut écrire sur le port correspondant 0x00.

2. LED-L

On déduit de ce qui précède des fonctions ledL_on et ledL_off ne prenant pas d'argument et permettant respectivement d'allumer et d'éteindre la LED avant gauche :

```
def ledL_on():
      pin8.write_digital(0x01)

def ledL_off():
      pin8.write_digital(0x00)
```

3. LED-R

De même, on obtient des fonctions ledR_on et ledR_off pour la LED avant droite :

```
def ledR_on():
      pin12.write_digital(0x01)

def ledR_off():
      pin12.write_digital(0x00)
```

4. LED-L et LED-R en simultané

Bien que cela ne soit pas nécessaire à l'allumage ou l'extinction simultanés des deux LEDs avant, on définit deux fonctions leds_on et leds_off ne prenant pas d'argument et permettant respectivement d'allumer les deux LEDs avant et de les éteindre.

```
def leds_on():
      ledL_on()
      ledR_on()

def leds_off():
      ledL_off()
      ledR_off()
```

V. Utilisation des LEDs situées sous le chassis

Sous le chassis se situent quatre LEDs RGB, toutes connectées au pin 15. Il est possible d'interagir avec ces LEDs via le module NeoPixel de Python, ces LEDs étant contrôlées par NeoPixel dans les exemples MakeCode fournis sur le site de DFRobot [3].
Afin de simplifier l'utilisation de ces LEDs pour les élèves, définissons une fonction ledrgb prenant en entrée :
· n : Entier entre 0 et 3 indiquant le numéro de la LED à allumer (les LEDs sont numérotées sous le châssis)
· r : Entier entre 0 et 255 correspondant à la valeur de la composante rouge du code RGB de la couleur à faire afficher par la LED.
· g : Entier entre 0 et 255 correspondant à la valeur de la composante verte du code RGB de la couleur à faire afficher par la LED.
· b : Entier entre 0 et 255 correspondant à la valeur de la composante bleue du code RGB de la couleur à faire afficher par la LED.

Et allumant la LED numéro n à la couleur définie par le code RGB (r,g,b).
On prendra garde à faire vérifier par le programme que les grandeurs entrées sont correctes : entières, et appartenant aux bons intervalles.

L'implémentation Python de la fonction ledrgb est :

```
from neopixel import NeoPixel
from math import *

def ledrgb(n,r,g,b):
    n=min(max(n,0),3)
    r=min(max(r,0),255)
    g=min(max(g,0),255)
    b=min(max(b,0),255)
    np=NeoPixel(pin15,4)
    np[floor(n)]=(floor(r),floor(g),floor(b))
    np.show()
```

VI. Utilisation des capteurs de suivi de ligne

1. Capteur de suivi de ligne gauche

Le capteur de suivi de ligne gauche correspond au pin 13 (cf. I.2). On en déduit une fonction lineL ne prenant pas d'arguments et qui renvoie True si ce capteur est dans l'état High, False sinon :

```
def lineL():
    if pin13.read_digital()==1:
        return True
    else:
        return False
```

Remarque : Le fait que le capteur de suivi de ligne gauche soit dans l'état High correspond à l'allumage d'une LED bleue sur l'avant gauche du robot (fig. 1)

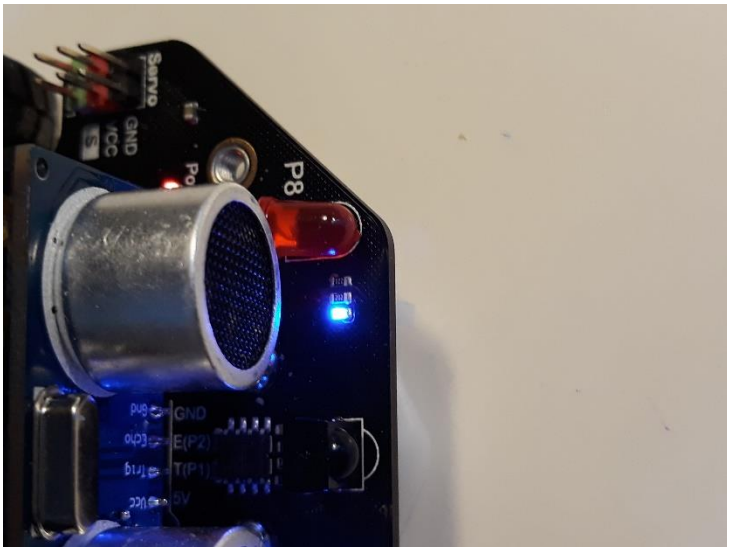


Fig. 1 : LED bleue témoignant de l'état High du capteur de suivi de ligne gauche.

2. Capteur de suivi de ligne droit

Mêmes remarques que dans la partie précédente, mais appliquées au côté droit du robot Maqueen. On en déduit une fonction lineR() :

```
def lineR():
      if pin14.read_digital()==1:
            return True
      else:
            return False
```

VII. Version finale de Pyqueen

Dans sa version 1.0, Pyqueen est donc :

```
#pyqueen v1.0 - 03/01/2020
#Copyright (C) 2020 Thomas Pochart
#This program is free software: you can redistribute it and/or modify it
under the terms of the GNU General Public License as published by the Free
Software Foundation, either version 3 of the License, or (at your option)
any later version.
#This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
for more details.
#You should have received a copy of the GNU General Public License along
with this program.  If not, see <https://www.gnu.org/licenses/>.

from microbit import *
from machine import *
from math import *
from neopixel import NeoPixel

addr=0x10

def motorL(n):
      if n>100:
            n=100
      if n<-100:
            n=-100
      if n>=0:
            direc=0x0
      else:
            direc=0x1
      v=abs(floor(n))*255//100
      i2c.write(addr,bytearray([0x0,direc,v]))

def motorR(n):
      if n>100:
            n=100
      if n<-100:
            n=-100
      if n>=0:
            direc=0x0
      else:
            direc=0x1
      v=abs(floor(n))*255//100
      i2c.write(addr,bytearray([0x2,direc,v]))

def motors(n):
      if n==0:
            f=0
      elif n>0:
            f=floor(4*n/5)+1
      else:
            f=-floor(-4*n/5)+1
      motorL(f)
      motorR(n)

def distance():
      if pin1.read_digital()==1:
```

```
            pin1.write_digital(0)
            sleep(10)
      pin1.write_digital(1)
      pin1.write_digital(0)
      pin2.read_digital()
      T=time_pulse_us(pin2, 1, 10000)
      if T<0:
            return(1000000)
      else:
            d=340*T/20000
            return(d)

def ledL_on():
      pin8.write_digital(0x01)

def ledL_off():
      pin8.write_digital(0x00)

def ledR_on():
      pin12.write_digital(0x01)

def ledR_off():
      pin12.write_digital(0x00)

def leds_on():
      ledL_on()
      ledR_on()

def leds_off():
      ledL_off()
      ledR_off()

def ledrgb(n,r,g,b):
    n=min(max(n,0),3)
    r=min(max(r,0),255)
    g=min(max(g,0),255)
    b=min(max(b,0),255)
    np=NeoPixel(pin15,4)
    np[floor(n)]=(floor(r),floor(g),floor(b))
    np.show()

def lineL():
    if pin13.read_digital()==1:
        return True
    else:
        return False

def lineR():
      if pin14.read_digital()==1:
            return True
      else:
            return False
```

Bibliographie

[1] I2c addresses and the BBC micro:bit
https://tech.microbit.org/hardware/i2c/
(Consultée le 31/12/2019)

[2] Maqueen MakeCode graphical library
https://github.com/DFRobot/pxt-maqueen
(Consultée le 31/12/2019)

[3] Maqueen programming example : RGB Breathing Ambient Light
https://makecode.microbit.org/v0/69610-55057-29165-54269
(Consultée le 31/12/2019)

[4] Thomas Pochart : Utilisation du robot micro:Maqueen v4.0 de DFRobot avec une carte BBC micro:bit sous Python, version 1.0