Comment un véhicule autonome peut-il effectuer un freinage d'urgence lorsqu'il rencontre un obstacle ?

SNT

Informatique embarquée et objets connectés

Véhicule autonome Maqueen

Capacités attendues	
Informatique embarquée et objets connectés	☑ Identifier des algorithmes de contrôle des comportements physiques à travers les données des capteurs, l'IHM et les actions des actionneurs dans des systèmes courants. ☑ Réaliser une IHM simple d'un objet connecté. ☑ Écrire des programmes simples d'acquisition de données ou de commande d'un actionneur.
Localisation	☐ Décoder une trame NMEA pour trouver des coordonnées géographiques.
Données structurées	☐ Identifier les principaux formats et représentations de données. ☐ Distinguer la valeur d'une donnée de son descripteur. ☐ Réaliser des opérations de recherche, filtre, tri ou calcul sur une ou plusieurs tables.

CONTEXTE

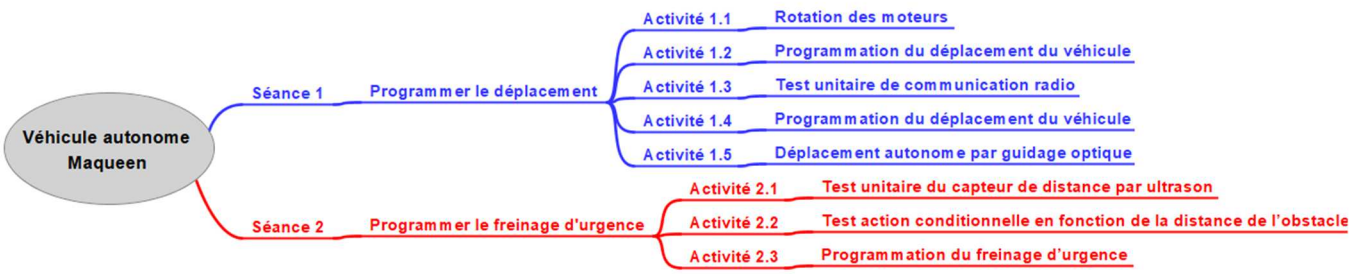
Le véhicule autonome est devenu un enjeu mondial, à la fois économique, écologique et de sécurité routière, aussi bien pour les pays constructeurs que pour les pays face à des défis de mobilité, notamment urbaine.

Aujourd'hui, malgré les avancées technologiques majeures, un grand nombre de défis restent encore à relever pour que les véhicules autonomes prennent place dans notre quotidien. L'avènement de la voiture « super-intelligente » ne pourra être que le fruit d'une collaboration entre de nombreux acteurs.

Membre d'une équipe de recherche sur la voiture autonome, vous avez en charge de mettre au point un programme de freinage d'urgence par détection d'obstacles.

DÉROULÉ

La séquence sur les maquettes de véhicules autonomes est proposée sur 2 séances de 1h30.



SITUATION DÉCLENCHANTE

Pour introduire la séquence, les élèves pourront visionner la vidéo suivante :

https://www.youtube.com/watch?v=S9HGO06dDHA

SEANCE 1

Question directrice : comment programmer le déplacement autonome véhicule ?

Activité 1.0 : Situation déclenchante

Vidéo : en 2018, le constructeur chinois Zhuzhou CRRC Times Electric a développé l'ART (Autonomous Rail Rapid Transit), un véhicule à mi-chemin entre le bus et le tramway, électrique et autonome, qui se déplace non seulement sans ligne aérienne mais également sans rails !

Mais comment fonctionne le système de guidage optique ?

Dans cette première partie, nous allons programmer le véhicule afin qu'il puisse se déplacer de manière autonome. Afin de concevoir correctement ce programme, nous allons procéder par étapes :

- Mise en rotation des roues (marche avant)
- Ajout des autres sens de déplacement (marche arrière et changement de direction)
- Essais du déplacement pas pilotage manuel (à partir d'une carte Micro:bit « télécommande »)
- Déplacement autonome par guidage optique (suiveur de ligne).

Rotation des moteurs

Activité 1.1 : rotation des moteurs

Capacité visée : Écrire des programmes simples d'acquisition de données ou de commande d'un actionneur.

À savoir :

Instruction pour piloter les moteurs du robot Maqueen :

```
from microbit import *

i2c.write(0x10, bytearray ([moteur, sens, vitesse])
```

moteur	sens	vitesse
Gauche = 0 Droit = 2	↻ = 0 ↺ = 1	Entre 0 et 255*

* Pour la vitesse, en dehors de ces nombres, un modulo est effectué.
Par exemple une vitesse de 300 vaut une vitesse de 45 et une vitesse de -45 vaut une vitesse de 210.

1.1.1- A l'aide de l'encadré ci-dessus, indiquez les deux instructions nécessaires pour mettre en rotation les deux moteurs du robot, vers l'avant, à la moitié de sa vitesse maximale.

1.1.2- Et indiquez les deux lignes d'instructions qui permettent de stopper le robot ?

1.1.3- En vous aidant de la base du programme ci-dessous, complétez les instructions afin que le robot avance tout droit (toujours à la moitié de sa vitesse maximale) pendant 2 secondes puis s'arrête.
(Voir fichier `1.1-Rotation_Moteur.py`)

Pensez à :
• Allumer l'interrupteur à l'arrière du véhicule Maqueen
• Brancher la carte du véhicule et cliquer sur « Flasher » pour téléverser le programme (si cela échoue, débrancher et rebrancher la carte puis flasher à nouveau).
• Appuyer si besoin sur le bouton « reset » de la micro:bit pour exécuter à nouveau le programme.

```
from microbit import *

# Définition des fonctions
def avance():
    i2c.write(0x10,bytearray([... , ... , ...])) #moteur droit, sens horaire, vitesse
    i2c.write(0x10,bytearray([... , ... , ...])) #moteur gauche, sens horaire, vitesse

def stop():
    i2c.write(0x10,bytearray([... , ... , ...])) # arret moteur droit,
    i2c.write(0x10,bytearray([... , ... , ...])) # arret moteur gauche

# Programme
avance()
sleep(2000)
stop()
```

Activité 1.2 : programmation du déplacement du véhicule

Nous allons maintenant ajouter les fonctions pour tourner et reculer.

1.2.1- En suivant la logique précédente, complétez le programme afin d'ajouter les trois fonctions `droite()`, `gauche()` et `recule()`.

Nota : pour tourner vers la droite, on stoppera la roue droite et on mettra en rotation uniquement la roue gauche… idem pour la gauche…

1.2.2-Tester les fonctions en créant un petit parcours de quelques secondes où le robot avance, tourne à droite, avance encore, tourne à gauche, recule puis stoppe (1 seconde par fonction).

Conclusion : Notre véhicule possède maintenant les fonctions de déplacement dans les 4 directions. Avant de pouvoir rendre le déplacement autonome, nous allons ajouter un moyen de pilotage manuel à distance.
Pour cela, une 2ème carte Micro:bit servira de télécommande et la carte du robot Maqueen devra être en mesure de recevoir le signal radio de la télécommande.

Activité 1.3 : test unitaire de communication radio

Implanter les deux programmes ci-dessous, respectivement sur la carte « télécommande » et sur la carte « robot Maqueen ».
(Voir fichiers `1.3-Test_radio_telecommande.py` et `1.3-Test_radio_maqueen.py`)

Dans le cas où il y a plusieurs robots Maqueen dans la classe, choisir une valeur de « group » différent des autres équipes (vous pouvez choisir un numéro de groupe entre 1 et 255)

```
# Pour carte télécommande

from microbit import *
import radio

radio.config (address=1, group=1)
radio.on()

while True :
    if button_a.is_pressed():
        radio.send ("A")
    else :
        radio.send ("-")
```

```
# Pour carte Robot Maqueen

from microbit import *
import radio

radio.config (address=1, group=1)
radio.on()

while True :
    donnees_reception = radio.receive ()
    if donnees_reception == "A":
        display.show (Image.SMILE)
    elif donnees_reception == "-":
        display.show (Image.SAD )
```

1.3.1- Modifier les programmes précédents afin d'allumer (bouton A) et d'éteindre (bouton B) à partir de la télécommande les LEDs (pin P8 et P12) du Robot Maqueen.

Activité 1.4 : programmation du déplacement du véhicule

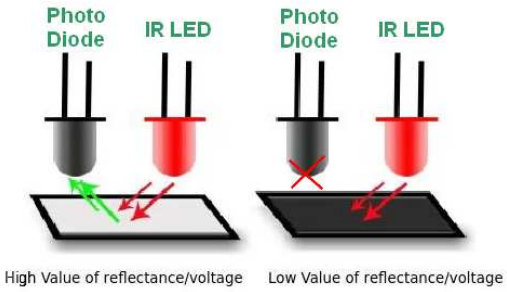
Capacité visée : Réaliser une IHM simple d'un objet connecté.

1.4.1- Compléter les programmes `1.4-Vehicule_radio_Telecommande.py` et `1.4-Vehicule_radio_Maqueen.py` **afin que le véhicule Maqueen se déplace suivant les mouvements de la carte télécommande (inclinaisons avant, arrière, droite et gauche via l'accéléromètre).**

Activité 1.5 : déplacement autonome par guidage optique

Capacité visée : Identifier des algorithmes de contrôle des comportements physiques à travers les données des capteurs, l'IHM et les actions des actionneurs dans des systèmes courants.

Principe des capteurs : le suiveur de ligne est composé de 2 capteurs à infrarouge Line-R (pin14) et Line-L (pin13) ayant chacun une LED émettrice infrarouge et une photodiode sensible. La LED émet un signal et la photodiode le détecte.
Le noir absorbant le rayonnement infrarouge, le récepteur ne détecte pas le signal émis lorsque le capteur est au-dessus de la ligne, celui-ci retourne donc 0. Dans le cas contraire, le signal est réfléchi par le blanc, le capteur retourne alors 1.

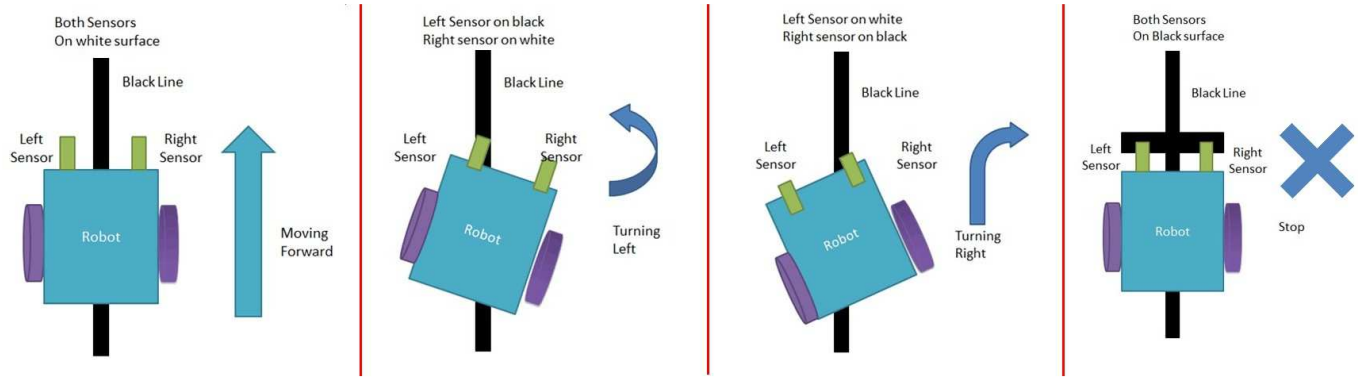


1.5.1- Tester le programme ci-dessous (voir le fichier `1.5-Guidage_optique.py`**)**
→ Demander une feuille A3 avec un circuit suiveur de ligne pour tester le programme.

```
from microbit import *

while True :
      if pin13.read_digital () == 0 and pin14.read_digital () == 0:
            i2c.write (0x10, bytearray ([0, 0, 30]))
            i2c.write (0x10, bytearray ([2, 0, 30]))
      if pin13.read_digital () == 0 and pin14.read_digital () == 1:
            i2c.write (0x10, bytearray ([0, 0, 0]))
            i2c.write (0x10, bytearray ([2, 0, 255]))
      if pin13.read_digital () == 1 and pin14.read_digital () == 0:
            i2c.write (0x10, bytearray ([0, 0, 255]))
            i2c.write (0x10, bytearray ([2, 0, 0]))
      if pin13.read_digital () == 1 and pin14.read_digital () == 1:
            i2c.write (0x10, bytearray ([0, 0, 0]))
            i2c.write (0x10, bytearray ([2, 0, 0]))
```

1.5.2- En utilisant l'image suivante expliciter le programme ci-dessus.



SEANCE 2

Question directrice : comment programmer le freinage d'urgence ?

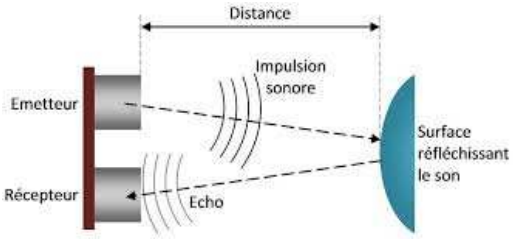
Activité 2.0 : Situation déclenchante

Maintenant que le véhicule se déplace de manière autonome, nous devons mettre au point un programme de freinage d'urgence du véhicule autonome si un obstacle est détecté devant le véhicule. Nous allons devoir procéder par étapes afin de concevoir correctement le programme.

Le freinage d'urgence devra être activé à partir d'une certaine distance critique de l'obstacle.
Il nous faut donc dans un premier temps être capable de récupérer la distance séparant le véhicule de l'obstacle.

Prérequis :

À savoir : le véhicule possède un **capteur de distance ultrason** : ce module émet un signal ultrason devant lui qui « rebondit » sur l'obstacle puis retourne au capteur qui « écoute » l'écho. En connaissant la vitesse de propagation du son, selon le temps que l'écho met pour revenir, on peut en déduire la distance de l'obstacle.



- Préparer le robot Maqueen (carte micro:bit enfichée boutons vers l'avant / capteur ultrason en place / 3 piles AAA / cordon USB…).
- Ouvrir votre logiciel de programmation compatible Micro:bit (MU editor…).

Activité 2.1 : Test unitaire du capteur de distance par ultrason

Capacité visée : Écrire des programmes simples d'acquisition de données ou de commande d'un actionneur.
Nous allons nous intéresser ici à la programmation du véhicule afin de récupérer sa distance avec des obstacles

• Ouvrez le programme suivant dans Mu Editor (`2.1-Test_ultrason.py`)

```
from microbit import *
import utime
import machine

def distance_cm() :
    v = 0.0340              # vitesse du son en micromètres par seconde
    pin1.write_digital(0)   # on met pin1 à l'état 0
    utime.sleep_us(2)       # pour éviter les interférences on attend 2 microsecondes

    pin1.write_digital(1)   # pin1 à l'é tat 1, une impulsion sonore est é mise
    utime.sleep_us(5)       # on attend 5 microsecondes
    pin1.write_digital(0)   # on cesse d'é mettre une impulsion sonore

    pin2.read_digital()     # pin2 en position d'écoute pour attendre le retour du signal
    temps = machine.time_pulse_us (pin2 , 1, 1000000)

    return temps *v/2 + 0.55 # on divise par 2 pour l'aller-retour, 0.55 dépend de l'étalonnage
                            # ce coefficient (0.55) varie légèrement d'une Micro:bit à l'autre

def affiche_distance():
    while True :
        print (distance_cm())
        sleep (500)
```

• Allumez l'interrupteur à l'arrière du véhicule Maqueen
• Branchez la carte du véhicule et clique sur « Flasher » (si cela échoue, débranchez et rebranchez la carte puis flashez à nouveau).
• Ouvrez la console de Mu Editor en cliquant sur REPL
• Exécutez le programme en saisissant « `affiche_distance()` » dans la console.

2.1.1- Testez le fonctionnement du capteur ultrason et expliquez ce que l'on constate

Activité 2.2 : Test action conditionnelle en fonction de la distance de l'obstacle

Dans cette partie, on souhaite qu'une action soit déclenchée en dessous d'une certaine distance mesurée par le capteur ultrason. Cela nous permettra de préparer le programme de freinage d'urgence.

2.2.1- Écrire un programme `(2.2-Ultrason_LED.py)` qui allume la LED « P8 » lorsque la distance est inférieure à 20 cm (et la laisse éteinte au-delà de 20cm)

Pour les plus avancés :

2.2.2- Écrire un programme `(2.2-Ultrason_RGB.py)` qui allume les LEDs RGB sous le robot en vert lorsque la distance est supérieure à 30 cm, en orange lorsque la distance est comprise entre 20 et 30 cm et en rouge si la distance est inférieure à 20 cm.

Activité 2.3 : Programmation du freinage d'urgence

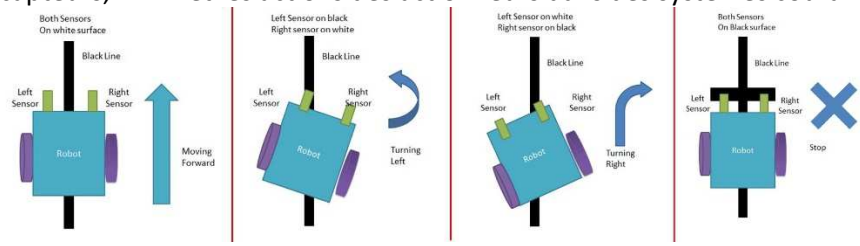
Capacité visée : Identifier des algorithmes de contrôle des comportements physiques à travers les données des capteurs, l'IHM et les actions des actionneurs dans des systèmes courants.

À présent, nous sommes prêts à implémenter le programme final qui arrêtera le véhicule (en marche autonome par guidage optique) s'il rencontre un obstacle.

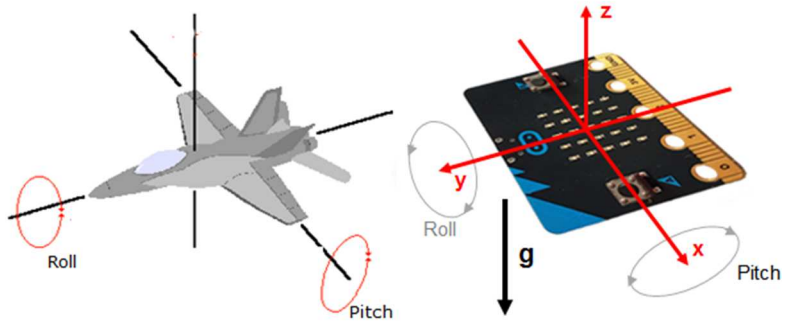
2.3.1- A partir des programmes précédents, concevoir le programme final qui stoppe le véhicule s'il est à moins de 10 cm d'un obstacle.

BILAN – SYNTHÈSE

Capacité visée : Identifier des algorithmes de contrôle des comportements physiques à travers les données des capteurs, l'IHM et les actions des actionneurs dans des systèmes courants.



Capacité visée : Réaliser une IHM simple d'un objet connecté.



Capacité visée : Écrire des programmes simples d'acquisition de données ou de commande d'un actionneur.