Périphériques d'entrée et de sortie
Interface Homme-Machine (IHM)

SIMULATION D'UN SYSTEME DE GESTION AUTOMATIQUE DE VEHICULE

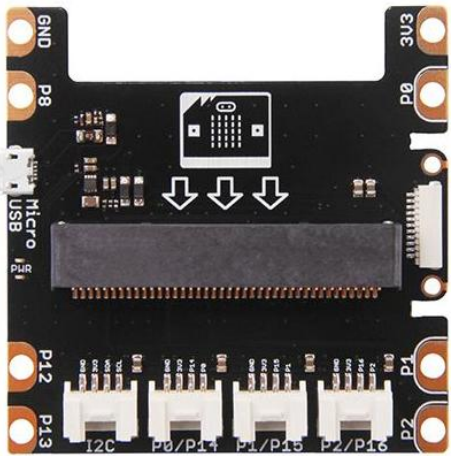
REFERENCE AU PROGRAMME

Niveau : 1ère N.S.I.

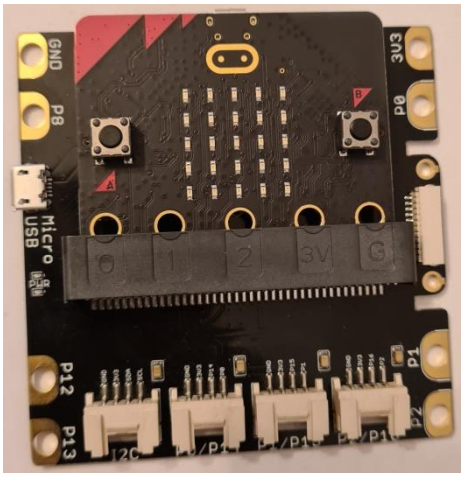
Contenus	Capacités attendues	Commentaires
Architectures matérielles et systèmes d'exploitation		
Périphériques d'entrée et de sortie Interface Homme Machine (IHM)	**Identifier le rôle des capteurs et actionneurs. Réaliser par programmation une IHM répondant à un cahier des charges donné.**	**Les activités peuvent être développées sur des objets connectés, des systèmes embarqués ou robots.**
Langages et programmation		
Constructions élémentaires	**Mettre en évidence un corpus de constructions élémentaires.**	**Séquences, affectation, conditionnelles, boucles bornées, boucles non bornées, appels de fonction.**
Utilisation de bibliothèques	**Utiliser la documentation d'une bibliothèque.**	**Aucune connaissance exhaustive d'une bibliothèque particulière n'est exigible.**

1. INTRODUCTION

Dans cette séance de travaux pratiques, nous allons utiliser la carte micro:bit et des périphériques. Pour cela nous aurons besoin d'un "Shield" compatible avec la technologie Grove (Plug and Play).



Insérer la carte `micro:bit` dans le shield mis à votre disposition en prenant garde au sens d'insertion (matrice à leds vers le haut).



2. RAPPEL DU FONCTIONNEMENT DU PILOTAGE DES ESSUIE-GLACES

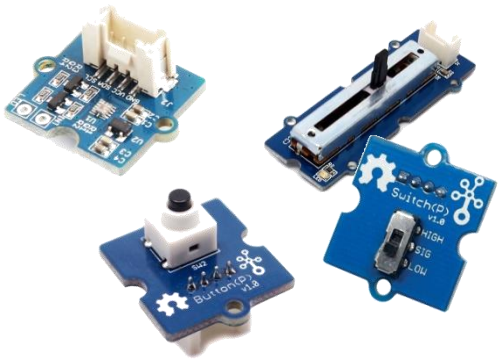
Fonctionnement du dispositif (cahier des charges)

Le dispositif permet d'améliorer le confort de conduite en automatisant la mise en service et l'arrêt des essuie-glaces.

Lorsque la clé de contact est tournée :

- Les essuie-glaces se mettent en marche :
  - dès que le système détecte suffisamment de pluie ; ils s'arrêtent quand la pluie cesse ;
  - par action sur le commodo d'essuie-glaces.

Constituants pour le prototypage



La fonction ACQUÉRIR est réalisée par les quatre éléments suivants :
un capteur de pluie simulé par un potentiomètre linéaire,
un interrupteur pour simuler le commodo d'essuie-glaces,
deux boutons poussoirs pour simuler le contacteur à clé (Neiman).

Les fonctions TRAITER, DISTRIBUER sont réalisées par :
une carte à microcontrôleur micro:bit,
une platine "BreadBoard".

Les fonctions CONVERTIR, TRANSMETTRE sont réalisées par :
un servo-moteur pour simuler les essuie-glaces.

La fonction COMMUNIQUER est éventuellement réalisée par :
une matrice à leds rouges pour indiquer que la clef de contact est tournée.

Note pour le TP

Les parties 3 à 6 sont indépendantes. Conserver vos sous-programmes (les mettre en commentaire) ; ils seront regroupés dans un deuxième temps.

3. MISE EN PLACE DU CONTACTEUR A CLE ET DES LEDS DE CONTROLE.

Rappel :

- Pour allumer des leds définies, créer une variable afin d'enregistrer une instanciation de l'objet `Image` puis appeler la méthode `show()` de l'objet `display` avec cette variable en paramètre.
  L'objet `Image` prend en paramètre une chaine de type :
  `'99999:00900:00900:00900:99900'`.
  Chaque élément séparé par "`:`" correspond physiquement à une ligne de leds.
- Pour écrire un texte à l'aide des leds, il faut appeler la méthode `show()` de l'objet `display` avec le texte à afficher en paramètre.

Exemple :

```
from microbit import *

while True:
    moon_image = Image('99999:\
                        00900:\
                        00900:\
                        00900:\
                        99900')
    display.show(mon_image)

    display.show('Bonjour')
```

3.1. Ouvrir le fichier sous Capytale, code : a46e-3517740

Algorithme du programme à implémenter et à téléverser dans la carte :

`Si le bouton "a" est appuyé :`
- `Afficher le message "ON" sur les leds (utiliser la méthode show() de l'objet display`
- `Afficher en permanence les trois leds centrales (utiliser la méthode show() de l'objet display`

`Si le bouton b est appuyé :`
- `Afficher le message "OFF" sur les leds`
- `Eteindre toutes les leds (display.clear())`

3.2. Écrire le programme, le tester sur la carte `micro:bit`.

Note : Conserver vos programmes au fur et à mesure. Les mettre en commentaire puis passer au chapitre suivant.

4. MISE EN ŒUVRE DES ESSUIE GLACE AVEC LE COMMODO.

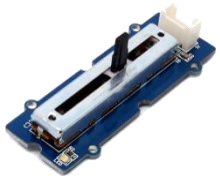
4.1. Connecter l'interrupteur fourni sur le port 0 du shield

Pour enregistrer les informations issues du capteur il faut utiliser la méthode `read_digital()` appliquée à l'objet `pin0` (numéro du port sur lequel est connecté le capteur).

```
commodo = pin0.read_digital()
```

4.2. Réaliser un programme permettant l'acquisition de la position du commodo.

5. GESTION DU NIVEAU DE PLUIE



5.1. Connecter le potentiomètre à glissière sur le port 1.

La carte `micro:bit` possède un convertisseur analogique – numérique (CAN) codé sur 10 bits.

Pour enregistrer les informations issues du capteur, utiliser la méthode `read_analog()` appliquée à l'objet `pin1` (numéro du port sur lequel est connecté le capteur).

```
pluie = pin1.read_analog()
```

5.2. Déterminer l'amplitude du mot binaire généré par le potentiomètre à glissière.

5.3. Créer une fonction `affiche_pluie()` qui prend en paramètre le mot binaire issu du potentiomètre à glissière `pluie` et qui réalise l'affichage du niveau de pluie sur la matrice à leds comme proposé dans l'algorithme ci-dessous.

```
Enregistrer la valeur issue du potentiomètre (intensité de la pluie)

fonction affiche_pluie(pluie)
     Si pluie est inférieur à 50,
          Ne rien afficher

     Si pluie est compris entre 50 et 125,
          Afficher la 1ère colonne de leds

     Si pluie est compris entre 125 et 256,
          Afficher les deux premières colonnes de leds

     Si pluie est compris entre 256 et 512,
          Afficher les trois premières colonnes de leds

     Si pluie est compris entre 512 et 768,
          Afficher les quatre premières colonnes de leds


     Sinon afficher toutes les leds.
```

5.4. Écrire le programme permettant d'enregistrer le mot binaire généré par le CAN et d'afficher le niveau de pluie sur la matrice à leds ainsi que dans la console (fonction `print()`) pour vérification du bon fonctionnement.

6. MISE EN PLACE DE L'ACTIONNEUR (SERVOMOTEUR)

Pour assurer la rotation du servomoteur, il existe deux techniques :
- ✓ la rotation continue,
- ✓ la rotation par unités d'angle.

Nous avons choisi la rotation continue. Pour information, la fonction permettant d'assurer la rotation selon des unités d'angle est également implémentée dans le fichier de travail.

Fonction de rotation continue

Pour imposer une direction et une vitesse au servomoteur, nous utiliserons la fonction `set_servo(pin, direction, vitesse)` qui prend en paramètres :
- ✓ le port de connexion `pin`,
- ✓ la direction souhaitée `direction` (`1` pour le sens antihoraire et `-1` pour le sens horaire),
- ✓ la vitesse désirée `vitesse` comprise entre `0` et `100`.

Cette fonction ne retourne rien.

Voici la fonction déjà implémentée dans le fichier :

```
def set_servo(pin, direction, vitesse):
  """ Fonction pour assurer la rotation du Servomoteur
      Paramètres d'entrée :
      - pin : variable indiquant le port de connexion (pin0, pin1 ou pin2)
      - direction : indique le sens de rotation (1 : antihoraire, -1 : horaire)
      - vitesse : vitesse de rotation comprise entre 0 et 100
  """
  if 0 <= vitesse <= 100:
    ecart = 14
    if direction == 1:
      vitesse_angle = 90 * (1 + vitesse / 100) - ecart
      pin.write_analog(vitesse_angle)
    elif direction == -1:
      vitesse_angle = 90 * (1 - vitesse / 100) - ecart
      if vitesse_angle < 0:
        vitesse_angle = 1
      pin.write_analog(vitesse_angle)
    else:
      raise ValueError("Servomoteur à rotation continue n'a pas de direction '" + str(direction) + "'")
  else:
    raise ValueError("Vitesse du servomoteur out of range '" + str(vitesse) + "'")
```

6.1. Connecter le servomoteur sur le port 2 du shield.

6.2. Tester la fonction en assurant le fonctionnement suivant :
- ✓ sens de rotation antihoraire avec une vitesse de 100,
- ✓ pause de 0,5 secondes,
- ✓ sens horaire de vitesse 100,
- ✓ pause d'une seconde,
- ✓ sens horaire (ou antihoraire) à vitesse nulle (permet de stopper le servomoteur et d'éviter sa surchauffe).

6.3. Écrire une fonction `rotation_servo(speed)` prenant en paramètre un entier `speed`, compris entre 0 et 5, indiquant le niveau de pluie (issue de la fonction `affichage_pluie`). Cette fonction assurera la rotation du servomoteur (même fonctionnement que précédemment) suivie d'une pause en fonction du niveau de pluie.

Exemple : si le niveau de pluie est de 2 (`speed == 2`) on provoque une rotation comme dans l'exercice 6.2. suivie d'une pause de 2 secondes.

Temps de pause entre 2 rotations :

Niveau de pluie (speed)	Temps de pause après rotation
1	3
2	2
3	1,5
4	1
5	0,5

7. REGROUPEMENT DES SOUS-PROGRAMMES

Nous allons à présent rassembler, en un seul programme, les diverses implémentations effectuées précédemment.

Rappel du fonctionnement désiré :

Boucle infinie
- ✓ si la clé de contact est tournée (après appui sur le bouton `a`),
  - ✓ afficher le message "`ON`",
  - ✓ mettre un drapeau (`flag` à `1`) pour mémoriser l'appui sur le bouton `a`,

  - ✓ tant que le `flag` est à `1`,
    - ✓ si le commodo d'essuie-glaces est activé,
      - Mettre les essuies glace sur la vitesse la plus rapide (fonction `rotation_servo`),

    - ✓ sinon,
      - relever le niveau de pluie,
      - afficher le niveau sur la matrice à leds (fonction `affiche_pluie`),
      - si le niveau de pluie est supérieur à 50,
        - mettre les essuie-glaces sur la vitesse adaptées (fonction `rotation_servo`),

    - ✓ si on appuie sur le bouton b,
      - affichage du message "OFF",
      - effacement de la matrice à leds,
      - mettre un drapeau (flag à 0) pour mémoriser l'appui sur le bouton b et sortir de la boucle,