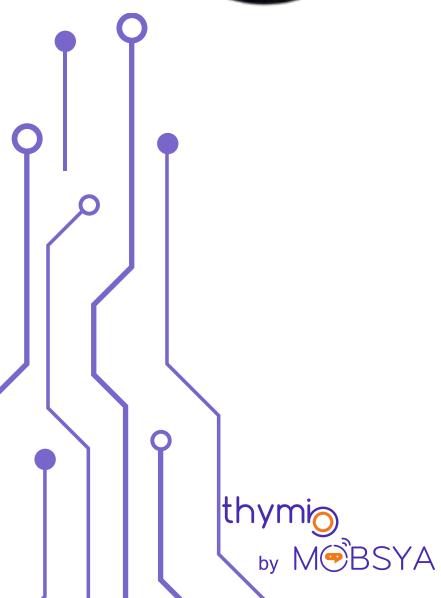


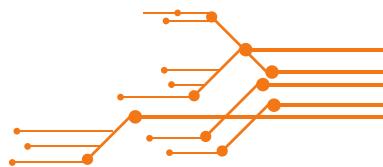
thymio

Programmer avec Python et Thonny



PAR: JOËL RIVET

Avant-propos



Après un peu plus d'une dizaine d'années d'existence, Thymio dans sa version 1 puis 2 s'est largement répandu dans le monde enseignant et associatif. Il doit ce succès à sa prise en main aisée, même de la part des plus jeunes, à la variétés de ses fonctionnalités, à sa programmation facile, à sa robustesse ainsi qu'à la large communauté de ses utilisateurs.

Néanmoins, Thymio est encore peu exploité dans le monde des lycéens et des élèves plus grands (disons au delà de quinze ans) ainsi qu' à l'université.

L' implantation du langage python dans l'écosystème de Thymio par l'EPFL (École Polytechnique Fédérale de Lausanne, Suisse), devrait permettre une plus large diffusion de ce robot dans ce monde d'élèves et de passionné.es plus agé.es.

Le cours que ce document présente a pour ambition de décrire l'ensemble des possibilités offertes par l' usage de python avec Thymio. Il est plus particulièrement destiné aux enseignants en informatique ou animateurs en associations.

Néanmoins, dans la plupart des cas, le contenu est parfaitement lisible par des élèves motivés.

Les enseignants seront à même de juger eux même les documents qu'ils pourront partager ou non avec les élèves.

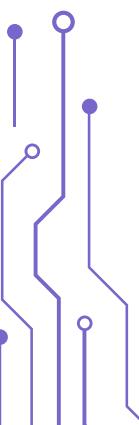
Ce cours est partagé en deux sessions :

- Une première session qui présente tout ce qui est nécessaire pour aborder Thymio-python.
 - Cinq chapitres de cours interactifs avec de petits scripts à étudier et à réaliser.
 - Une série de dix activités avec sujets, aides et solutions qui montrent que Thymio est capable d'effectuer une grande variété de tâches, ludiques ou plus sérieuses.
 - Une référence de l'API python pour Thymio
- Une deuxième session, plus avancée, qui présente des aspects moins connus de l'écosystème Thymio, les particularités de python appliquée à Thymio, ainsi qu'une quinzaine d'activités très variées qui pourront éventuellement constituer des points de départ pour des projets plus ambitieux, pour le baccalauréat par exemple.

Afin de profiter au mieux de ce cours, il faut être un familier de la programmation et connaître les éléments de base de python.

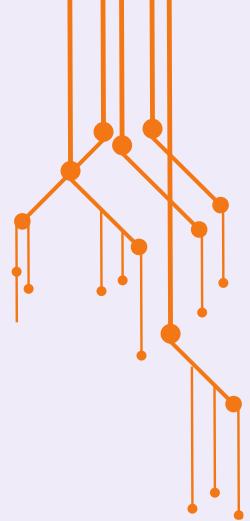
Il est également souhaitable, sans que cela soit du tout indispensable, de connaître les principes de base de la robotique et de Thymio.

Je forme le plaisir que j'ai eu à développer ce cours sera suffisamment contagieux pour que vous puissiez, cher lecteur, profiter à votre tour de ce robot étonnant qu'est Thymio II.



Auteur: JOËL RIVET

JOËL RIVET



SESSION 1

CHAPITRE 1

Thymio est un robot

CHAPITRE 2

Python à Thymio



CHAPITRE 3

Thymio Suite et Thonny

CHAPITRE 4

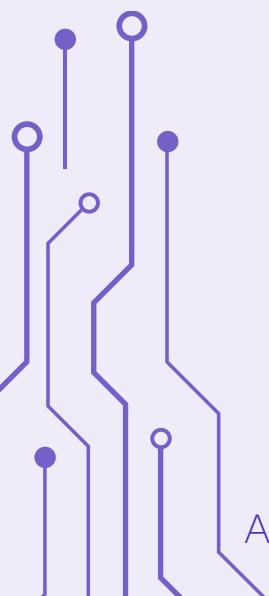
Programmer capteurs
et actionneurs - 1

CHAPITRE 5

Programmer capteurs
et actionneurs - 2

ANNEXE

API Thymio Python



Auteur: JOËL RIVET

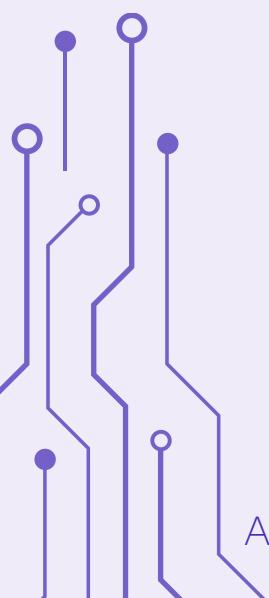


SESSION 1

CHAPITRE 1

Thymio est un robot

P1 - P4



Auteur: JOËL RIVET

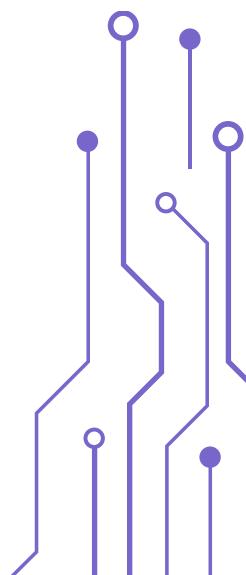
CONTENTS

01

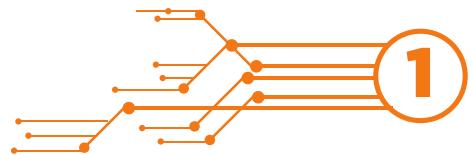
ROBOTIQUE - NOTIONS
FONDAMENTALES

02

LE ROBOT THYMIO II



Robotique - Notions fondamentales



Tout le monde ne s'accorde pas sur la définition d'un robot. Nous prendrons le parti d'en définir les contours par des notions très concrètes.



aspirateur domestique
(source pixabay)



drone (source pixabay)

A gauche, un aspirateur domestique autonome est conçu pour aspirer la poussière en se déplaçant dans un environnement changeant. Il gère les obstacles et leurs trajectoires grâce à ses capteurs embarqués et certains peuvent créer une carte de la pièce grâce à des télémètres laser.

A droite, une drone évolue dans l'air grâce à la commande de son utilisateur. Les drones bon marché sont rarement équipé de capteurs. Ils peuvent avoir une caméra mais ne s'en servent pas pour leur auto-navigation.
Sont ce tous deux des robots ?

Un robot est un appareil électro-mécanique qui comporte :

- des capteurs : dispositifs capables de récupérer des informations physiques sur son environnement.
Exemples : caméra, radar, détecteur infrarouge, capteur sonore, accéléromètre, thermomètre
- des actionneurs : dispositifs, souvent électro-mécaniques, rendant le robot capable d'effectuer certaines actions. Exemples : moteur pour se déplacer, bras manipulateur, LEDs, émetteur sonore ...

- un processeur (micro-contrôleur) : dispositif capable d'analyser les données des capteurs et de commander les actionneurs en conséquence.
- des programmes : rédigé par un humain. Il consiste à indiquer au processeur les commandes qu'il doit communiquer aux actionneurs du robot. Ces commandes dépendent bien sûr du comportement attendu par le programmeur.

On voit bien que l'aspirateur décrit est un robot, et que le drone n'en est pas un. C'est simplement une machine télécommandée.

Boucle sensori-motrice

Les données captées et les actions s'enchaînent dans un boucle qualifiée de **boucle sensori-motrice**.

On parle de boucle car le processeur scrute en permanence les données des capteurs de manière cyclique.

Par exemple, imaginons un robot à l'arrêt dont le capteur avant détecte un obstacle. Le programme pourra décider que tout mouvement de l'obstacle donnant lieu à une modification des valeurs du capteur produit un **événement**. Le processeur pourra décider d'une action en réponse à cet événement. Par exemple, reculer si l'obstacle avance.



Le robot Thymio II

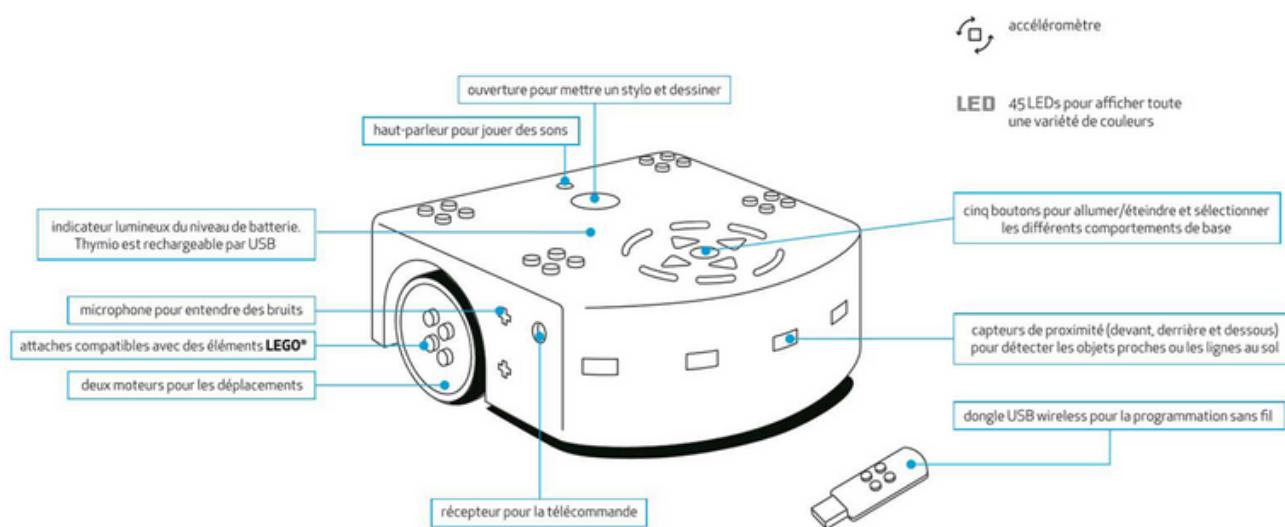


Thymio est un robot à roues conçu par l'EPFL (École Polytechnique Fédérale de Lausanne) il y a un peu plus d'une dizaine d'années. Son développement est assuré depuis par l'association Mobsya situé à Lausanne en Suisse.

Il est assez répandu dans l'enseignement, notamment auprès des plus jeunes.

Il est de conception robuste, open-source, très complet en terme de fonctionnalités et de prix relativement modéré.

La figure ci - dessous résume l'ensemble de ses capteurs et actionneurs.



Nativement, il est programmé dans le langage Aseba, un langage aux fonctionnalités restreintes qui s'adapte parfaitement à un système embarqué disposant d'un micro-contrôleur de capacité limitée.

Pour faciliter sa programmation, une famille d'interfaces a été développée, permettant sa programmation avec divers langages.

- des langages visuels comme VPL, blockly ou scratch
- des langages textuels comme python ou aseba.



Auteur: JOËL RIVET

Chapitre 1 - Thymio est un robot



L'utilisateur aura ainsi le choix entre trois façons d'accéder à ces langages :

- le logiciel (à installer) Thymio Suite qui donne un accès facile à tous ces langages.
- le site en ligne Vittascience qui propose une programmation python à l'aide de blocs de la technologie blockly.
- La plateforme lab.open-roberta
- Jupyter note-book pour une programmation en python et aseba (utilisateurs avancés).

A ces langages s'ajoutent deux accès :

- **Programmes pré-installés** : Par défaut et sans connexion, Thymio possède des programmes pré-installés opérationnels, qui procurent à Thymio six comportements particuliers. Ces comportements permettent une approche aisée de la robotique par les élèves les plus jeunes.
- **Intelligence artificielle** : Thymio peut être contrôlé par des algorithmes neuronaux d'intelligence artificielle via le logiciel alphai, développé par la société Learning Robots.

Le tableau ci-dessous résume les interfaces disponibles dans Thymio Suite.

Les langages de Thymio						
Langage	Comportements de base	VPL & VPL3	Blockly	Scratch	Aseba & Python	ThymioAI
Interface	Le Robot	Visuel (carte)	Visuel (block)	Visuel (block)	Textuel	Textuel
Langage natif	Aseba	Aseba	Javascript	Javascript	Aseba	Python
Liaison permanente	non	non	non	oui	non	oui
Type	Événementiel	Événementiel	Événementiel	Événementiel & séquentiel	Événementiel	Événementiel
Âge recommandé	5 ans	5 ans	7 ans	11 ans	15 ans	-
Difficulté	Très facile	Très facile	Facile	Moynenne	Avancé	-



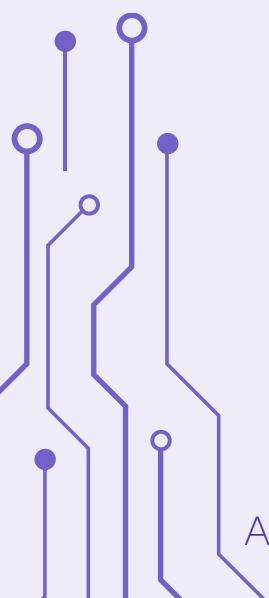


SESSION 1

CHAPITRE 2

Python à Thymio

P5 - P6



Auteur: JOËL RIVET

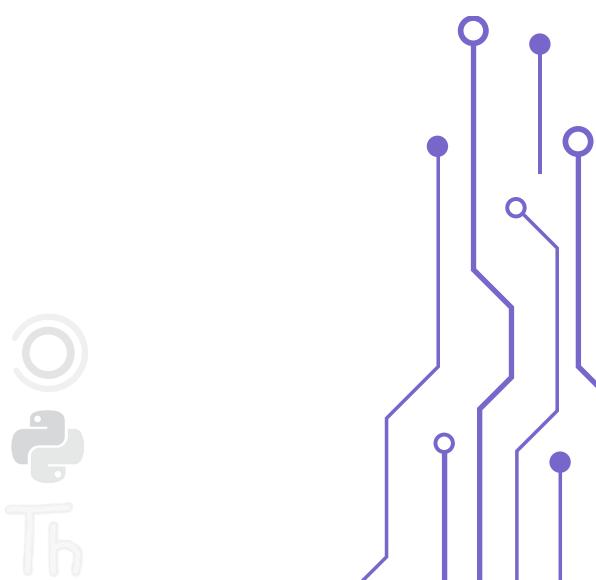
CONTENTS

01

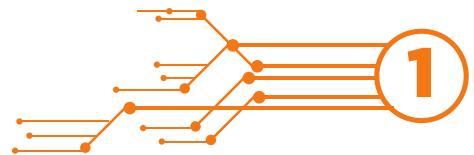
ASEBA, LANGAGE NATIF
DE THYMIO

02

DE PYTHON À ASEBA



Aseba, langage natif de Thymio



A l'intérieur du robot

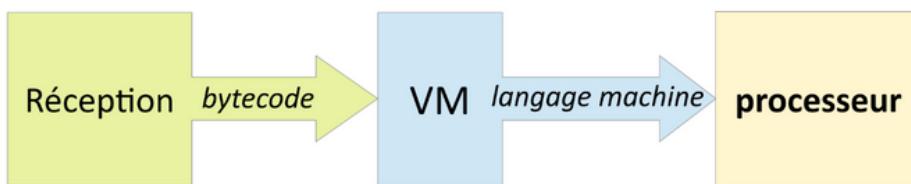
Thymio possède un micro-contrôleur (incluant un processeur) dont la tâche principale est d'exécuter le programme chargé dans sa mémoire, lui permettant ainsi de contrôler le robot.

Le code exécuté par le processeur est désigné par l'expression de *langage machine*.

Mais le code du programme n'est pas envoyé sous cette forme au robot.

Il est envoyé sous forme de *bytecode*, un format plus souple que le *langage machine*. Ce *bytecode* est interprété par un programme appelé **Virtual Machine** (VM) qui le convertit en *langage machine*.

La figure ci-dessous résume le chemin du code.



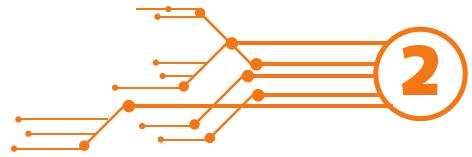
Aseba et le bytecode

Aseba est un langage texte élaboré essentiellement pour Thymio. C'est un langage aux fonctionnalités restreintes conçu pour un micro-contrôleur de capacité mémoire et de calcul limitées.

Ce langage réside dans l'ordinateur de l'utilisateur et non dans le robot.

Le script Aseba (code sous forme de texte) est compilé en *bytecode* par un programme appelé **Thymio Device Manager** (TDM). Puis le *bytecode* est envoyé au robot par un câble USB ou des ondes radio via le dongle USB wireless.

Python et Aseba

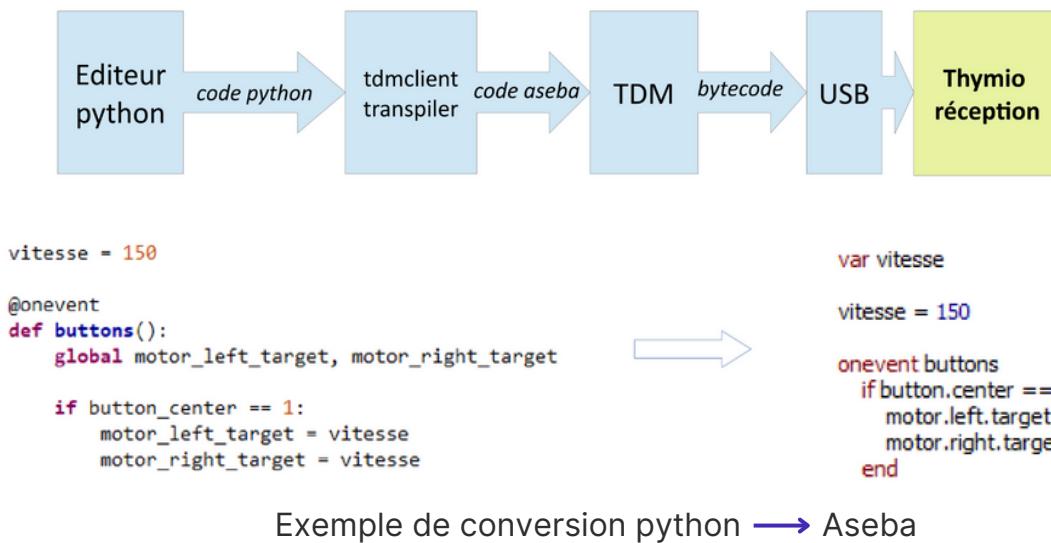


tdmclient

Programmer Thymio avec le langage python implique de pouvoir convertir le code (texte) python en code Aseba (texte aussi). Un module python appelé **tdmclient** a été développé dans ce but. Ce module permet entre autre de :

- accéder aux variables du robot
- transpiler le code python en code aseba (transpiler : reécrire les instructions python en instructions aseba)
- envoyer le code aseba au TDM (qui l'envoie au robot)

Pour en savoir plus, voir <https://pypi.org/project/tdmclient/>



Thymio Suite et Thonny

Pour saisir le code python, plusieurs possibilités sont offertes :

- via Thymio Suite <https://www.thymio.org/fr/telecharger-thymio-suite/>
- en mode console
- via jupyter notebook

Nous retiendrons par la suite la première option, Thymio Suite, qui ne nécessite qu'une seule installation et qui permet également d'accéder à tous les autres langages.

Pour les deux autres modes, référer à <https://pypi.org/project/tdmclient/>

Thymio Suite est un logiciel qui intègre le module tdm pour la connexion aux robots et permet de choisir une des six interfaces de programmation : VPL, VPL3, Blockly, Scratch, Python et Aseba.

L'interface python est assuré par l'éditeur Thonny, associé à un plug-in tdmclient.ty.

Le chapitre 3 détaille l'utilisation de Thonny et l'implémentation de python pour aseba.



Auteur: JOËL RIVET

Chapitre 2 - De python à Thymio



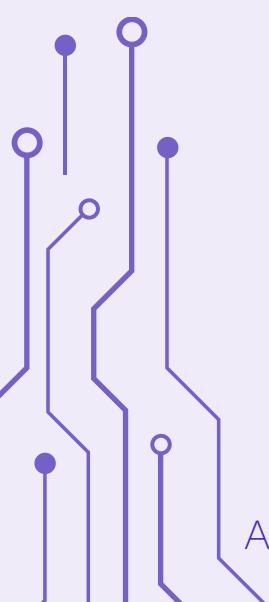


SESSION 1

CHAPITRE 3

Thymio Suite et Thonny

P7 - P17



Auteur: JOËL RIVET

CONTENTS

01

PRÉSENTATION DE
THYMIO SUITE

02

THONNY, UNE INTERFACE
DE PROGRAMMATION
PYTHON

03

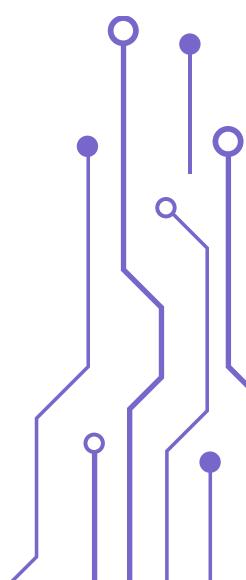
UN PREMIER
PROGRAMME

04

LE SIMULATEUR

05

ANNEXE : CAPTEURS ET
ACTIONNEURS COMPATIBLES
SIMULATEUR



Présentation de Thymio Suite



1.1 Thymio Suite

Thymio Suite est disponible en téléchargement gratuit et utilisable sur PC, Mac, Linux et tablette.

Thymio Suite porte deux fonctionnalités principales :

- lancer le programme TDM (voir chapitre 2) qui établit la connexion entre le ou les robots, et l'ordinateur de l'utilisateur.
- Proposer des interfaces de programmation pour les langages disponibles (pour une vue d'ensemble des interfaces, voir le tableau du chapitre 1)

Dans le cas où l'utilisateur ne possède pas de Thymio, ou pour tester des configurations particulières, on peut lancer un simulateur 3D (voir §4).

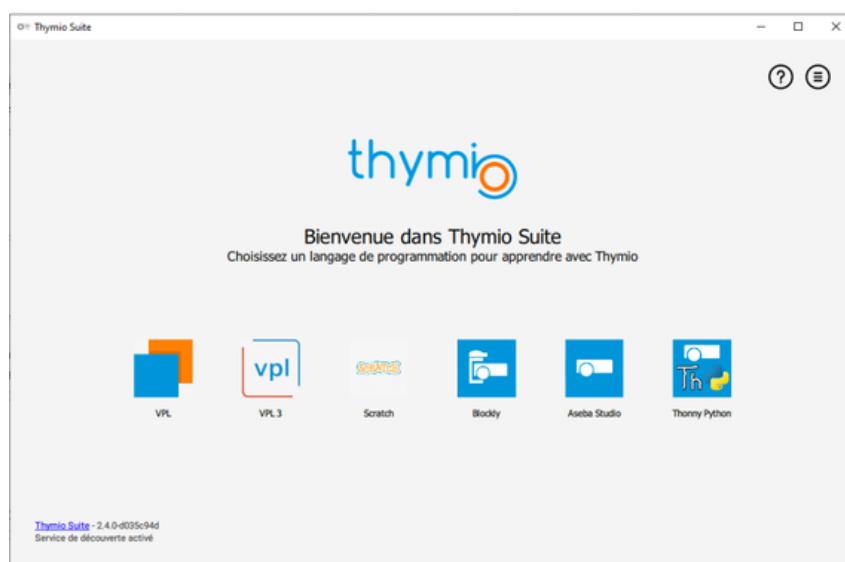
1.2 Téléchargement, installation, connexion

Télécharger gratuitement Thymio Suite à l'adresse:

<https://www.thymio.org/fr/telecharger-thymio-suite/>

- Installer le logiciel (en s'assurant que vous disposez des droits administrateur).
- Connecter un Thymio à l'ordinateur
 - soit par le câble USB (le robot s'allume tout seul).
 - soit par la clé USB sans fil. Dans ce cas, allumer le robot en appuyant sur le bouton central pendant quelques secondes. Vous devriez voir une LED clignoter sur la clé et sur le robot, signe que la connexion par le TDM est bien établi.

1.3 L'écran d'accueil



Auteur: JOËL RIVET

Chapitre 3 - Thymio Suite et Thonny

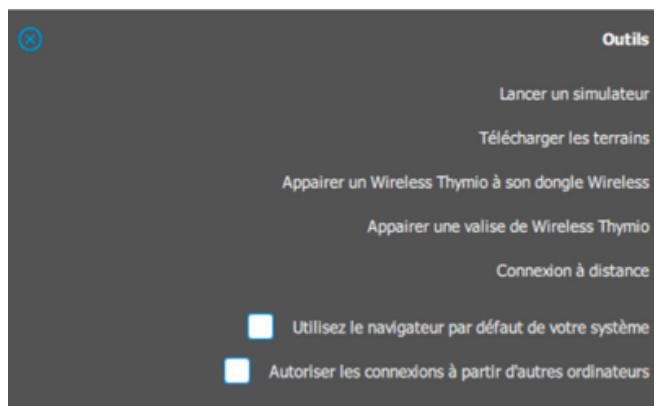


En cliquant sur une des 6 icônes, on accède à une interface de programmation.
Rappelons brièvement les interfaces.

- VPL propose une programmation à l'aide cartes à déplacer, programmation très adapté aux débutants et aux plus jeunes pour découvrir Thymio. VPL3 est une extension plus puissante de VPL
- Scratch, bien connu des collégiens, étend les fonctionnalités en introduisant notamment la possibilité de faire de la programmation séquentielle.
- Blockly est une autre interface à base de blocs événementiels, qui reste donc plus proche de VPL et d'Aseba que scratch.
- Avec Aseba Studio, on programme en mode texte en Aseba
- Thonny python permet de programmer en python.

Il y a 3 autres icônes cliquables :

- l'une en bas à gauche [Thymio Suite](#) pour accéder au site web de Thymio.
- la classique icône  d'aide en ligne en haut à droite.
- l'icône  qui ouvre un menu qui comporte des options :

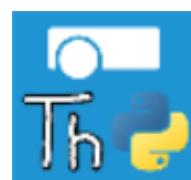


- La première option permet de lancer le simulateur
- La deuxième est surtout destinée aux utilisateurs de Mac pour lesquels la première option ne propose pas de terrains.

Nous ne détaillons pas les suivantes

1.4 Accéder à Thonny

Cliquez sur l'icône  Dans la fenêtre qui s'ouvre, il faut tout d'abord sélectionner le robot que l'on veut programmer en cliquant sur son icône.



Auteur: JOËL RIVET

Chapitre 3 - Thymio Suite et Thonny



Sur la figure ci-contre 2 robots apparaissent.

- Celui de gauche est un robot réel nommé 7011.
- Le robot de droite est un robot du simulateur, qui a donc été ouvert précédemment.



Sélectionnez un des 2 robots puis cliquez sur le bouton:

Programmer avec Thonny Python.

La fenêtre Thonny s'ouvre

Thonny, une interface de programmation python



2

2.1 Description de l'interface

Il s'agit dans ce paragraphe de décrire les fonctionnalités principales pour programmer Thymio. Vous pouvez voir que l'interface présente un programme qui sera abordé dans le paragraphe suivant.

C'est une interface assez classique d'IDE, la légende de la figure ci-dessous parle d'elle-même.

- Le menu Fichier et les 3 premières icônes permettent d'ouvrir et d'enregistrer les programmes.
- Les 5 icônes suivantes ne sont pas utiles dans notre contexte.
- Les 2 icônes permettent de lancer et d'arrêter le programme en cours sur le robot sélectionné à droite.

Une option Thymio est ajouté à la barre de menu classique de Thonny :

- un bouton run qui lance l'exécution du programme.
- L'option Transpile Program qui affiche dans la console le code Aseba issu du transpilage du code python.
- Le bouton stop qui arrête l'exécution du programme sur Thymio.
- L'option Unlock Thymio permet de rendre le robot disponible pour un autre langage.



Auteur: JOËL RIVET

Chapitre 3 - Thymio Suite et Thonny



Thonny - F:\Activités_pedagogiques\Robotique-Thymio\python\programmes perso\hangar\chapitre-3.py @ 40:25

Fichier Édition Affichage Exécuter Outils Aide Thymio

Pour lancer ou arrêter un programme

Cadre de saisie et d'édition du code

```

chapitre-3.py
1 vitesse = 0
2 nf_leds_top(0,0,0)
3 abandon = False
4
5 @onevent
6 def buttons():
7     global motor_left_target, motor_right_target, vitesse
8
9     if button_center == 1:
10        vitesse = 150
11        motor_left_target = vitesse
12        motor_right_target = vitesse
13        nf_leds_top(0,32,0)
14
15 @onevent
16 def prox():
17     global motor_left_target, motor_right_target, timer_period
18
19     if abandon: return
20
21     if prox_horizontal[2] > 3500:
22         motor_left_target = 0
23         motor_right_target = 0
24         nf_leds_top(0,32,0)
25         timer_period[0] = 2000
26     else:
27         motor_left_target = vitesse
28         motor_right_target = vitesse
29         nf_leds_top(0,32,0)
30         timer_period[0] = 0
31
32 @onevent
33 def timer0():
34     global motor_left_target, motor_right_target, timer_period, abandon
35
36     abandon = True
37     motor_left_target = -vitesse
38     motor_right_target = -vitesse
39     nf_leds_top(0,0,32)
40     #time.sleep(0.1) -- a

```

Console

Console qui permet d'afficher les erreurs, le résultat de la fonction print() et où l'on peut saisir du code python

Variables Aide Notes

Assistant Thymio Robots

Name	Status
Thymio 0 sur Bureau3 - 14	available
✓ 7011	ready

Pour sélectionner les robots disponibles (cliquer sur le nom)

2.2 Plusieurs robots et plusieurs programmes

Plusieurs robots

Comme le suggère la figure ci-contre de l'éditeur, il est possible de programmer plusieurs robots à partir d'une seule interface. On remarque :

Assistant x Thymio Robots x	
Name	Status
Thymio 0 sur Bureau3 - 14	available
✓ 7011	ready

- Le statut **ready** et la coche devant le robot 7011. C'est ce robot qui recevra le code quand l'utilisateur cliquera sur le bouton **run** .
- le statut **available** indique que ce robot est disponible (connecté) et donc utilisable par un des 6 programmes de Thymio Suite.

Plusieurs programmes

Pour montrer que l'on peut programmer plusieurs langages à la fois avec Thymio Suite, programmons avec le langage VPL le robot du simulateur (Thymio 0 sur Bureau 3 dans l'exemple, mais ce nom change avec chaque ordinateur).



Auteur: JOËL RIVET

Chapitre 3 - Thymio Suite et Thonny

- Ne pas fermer la fenêtre Thonny.
- Fermer la page Thonny Python de la fenêtre Thymio Suite avec le bouton  en haut à droite. La fenêtre d'accueil de Thymio Suite apparaît.
- Ouvrir une fenêtre VPL. Puis choisir le robot du simulateur.
- Ouvrir VPL.
- Revenir dans la fenêtre Thonny Python. On constate que le robot du simulateur n'est plus disponible, il est marqué **busy** (occupé en anglais).

On peut donc programmer plusieurs robots, dans des langages différents, à partir d'une seule instance de Thymio suite.

Il est même possible de prendre en charge des robots connectés à d'autres ordinateurs d'un même réseau local ou d'un réseau distant!

Voir les options Connexion à distance et Autoriser les connexions à partir d'autres ordinateurs du menu  de l'écran d'accueil de Thymio Suite (§1.3).

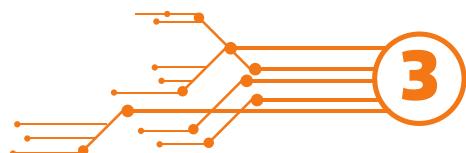
Si vous désirez programmer plusieurs robots avec le même éditeur, il suffit de cliquer sur le nom du robot choisi.

ATTENTION, un programme n'est pas spécifiquement attaché à un robot. Avant chaque envoi de programme avec run, il faudra s'assurer que c'est le bon robot qui est coché.

Subtilité avec le bouton stop  . Cliquer sur ce bouton a pour action de vider le robot du programme chargé.

Donc, si on clique sur stop, le bouton n'agit pas sur le programme visible dans l'onglet de droite, mais sur le robot **sélectionné à droite** !

Un premier programme



Généralement, dans les présentations de programmation, on commence par un tout petit programme, du genre "Hello, world !", pour mettre à l'aise le lecteur.

Ici, nous verrons un peu plus grand, mais, soyez rassuré, les chapitres suivants reprendrons tout en détail .

Le programme que l'on va étudier à présent permet de bien appréhender le paradigme de la programmation événementielle. Cette manière d'aborder la programmation (paradigme) n'est familière qu' à ceux qui ont programmé des interfaces web ou des gui (graphic user interface). Ce paradigme a tendance à dérouter parfois ceux qui n'ont pas programmé d'interface graphique.



En robotique, la plupart du temps, la programmation est uniquement événementielle, sans présence de code séquentiel.

Nous ne détaillerons pas ici toutes les instructions. Les fonctions et variables associées au robot seront décrites de manière progressive dans les chapitres 4 et 5.

Voici le programme :

Ce programme figure aussi dans un document appelé *code python - demonstation.pdf* de sorte que vous puissiez le consulter dans une fenêtre séparée tout en suivant ce document.
Pour des raisons de portabilité, les noms de variables seront en anglais dans tous les exemples et exercices du cours.

```
speed = 0
nf_leds_top(0,0,0)
cancel = False

@onevent
def buttons():
    global motor_left_target, motor_right_target, speed
    if button_center == 1:
        speed = 150
        motor_left_target = speed
        motor_right_target = speed
        nf_leds_top(0,32,0)

@onevent
def prox():
    global motor_left_target, motor_right_target, timer_period

    if cancel: return

    if prox_horizontal[2] > 3500:
        motor_left_target = 0
        motor_right_target = 0
        nf_leds_top(0,32,0)
        timer_period[0] = 2000
    else:
        motor_left_target = speed
        motor_right_target = speed
        nf_leds_top(0,32,0)
        timer_period[0] = 0

@onevent
def timer0():
    global motor_left_target, motor_right_target, timer_period, cancel

    cancel = True
    motor_left_target = -speed
    motor_right_target = -speed
    nf_leds_top(0,0,32)
    timer_period[0] == 0
```



Auteur: JOËL RIVET

Chapitre 3 - Thymio Suite et Thonny



Structure du code

On observe 4 parties :

- les 3 premières lignes créent et initialisent des variables
- La fonction `nf_leds_top(0,0,0)` éteint les LEDs de dessus
- puis 3 blocs de 3 fonctions commençant par le décorateur `@`. Chacune des 3 lignes `@onevent` permet de référencer des fonctions de sorte que la fonction s'exécute quand un événement particulier est détecté.

La consultation des données

Le processeur dans le robot reçoit en permanence les données issues des capteurs du robot. Dans un même temps, le code est parcouru pour déterminer si les données reçues doivent être prises en compte et déclencher l'exécution d'une des fonctions.

Par exemple, les données sur les boutons (Buttons), c'est-à-dire l'appui sur un des boutons ou non sont consultées 20 fois par seconde (20Hz). La fonction **buttons()** sera donc exécutée 20 fois par seconde.

Pour la fonction **prox()** qui correspond aux capteurs infrarouge détectant les obstacles, la fréquence est de 10 fois par seconde.

Comportement du code

La fonction **buttons()** commence par rendre global toutes les variables qui seront modifiées dans la fonction, que ce soit des variables créées par l'utilisateur (`speed`) ou le module `tdmclient`, `motor_left_target`.

20 fois par seconde, le code de cette fonction est parcourue. La plupart du temps, le bouton central ne subit aucun appui, la variable `button_center` reste alors à 0.

Si l'utilisateur a appuyé sur le bouton central, la variable `button_center` passe à 1. Le test `if button_center == 1:` est vrai, le code associé s'exécute :

- `speed = 150`
- `motor_left_target = speed` et `motor_right_target = speed`. Les valeurs de ces variables sont passés aux moteurs qui font démarrer Thymio.
- `nf_leds_top(0,32,0)` : fonction qui allume les LEDs du dessus en vert.



Auteur: JOËL RIVET

Chapitre 3 - Thymio Suite et Thonny



Comportement global du programme

Peut être serez à même de décrire ce que fait ce programme ?

En résumé, voici son comportement :

Une fois le programme lancé, Thymio est immobile (vitesse à zéro).

Si on appuie sur le bouton central, il démarre.

Si un objet se trouve devant lui (`prox_horizontal[2] > 3500`) , il s'arrête (`motor_left_target = 0, motor_right_target = 0`) et passe en rouge (`nf_leds_top(0,32,0)`)

La variable `timer_period[0]` prend la valeur 2000. Ça déclenche un minuteur réglé à 2000 ms ou 2s.

Le temps du minuteur s'écoule en mode asynchrone, indépendamment du reste du code.

Une fois le temps échu (au bout de 2s), la fonction `timer0` s'exécute.

Si on retire l'objet avant l'échéance de 2 secondes, le bloc de la clause `else` de la fonction `prox` s'exécute.

Le robot roule à nouveau (en rouge) tandis que la variable `timer_period[0]` est mise à zéro, ce qui a pour effet d'arrêter le minuteur. Le robot poursuit donc sa progression.

Si l'objet n'est pas retiré, au bout de 2 s, la fonction `timer0` s'exécute, le robot recule, (`motor_left_target = -speed,`

`motor_right_target = -speed`) en bleu et le minuteur est arrêté (si on ne l'arrête pas, il reprend son décompte de 2s).

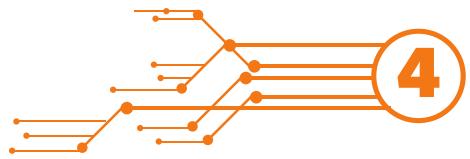
Important, la variable `cancel` est mise à vrai. Donc au cours du parcours du code, le code de la fonction `prox` sera ignoré à cause du test `if cancel: return`.

En résumé, le robot avance et s'arrête si il y a un obstacle. Si l'obstacle n'est pas retiré assez vite, le robot renonce et recule.

Si vous trouvez ce programme difficile à comprendre, ne soyez pas inquiet ! La suite sera plus progressive en reprenant tous des éléments.



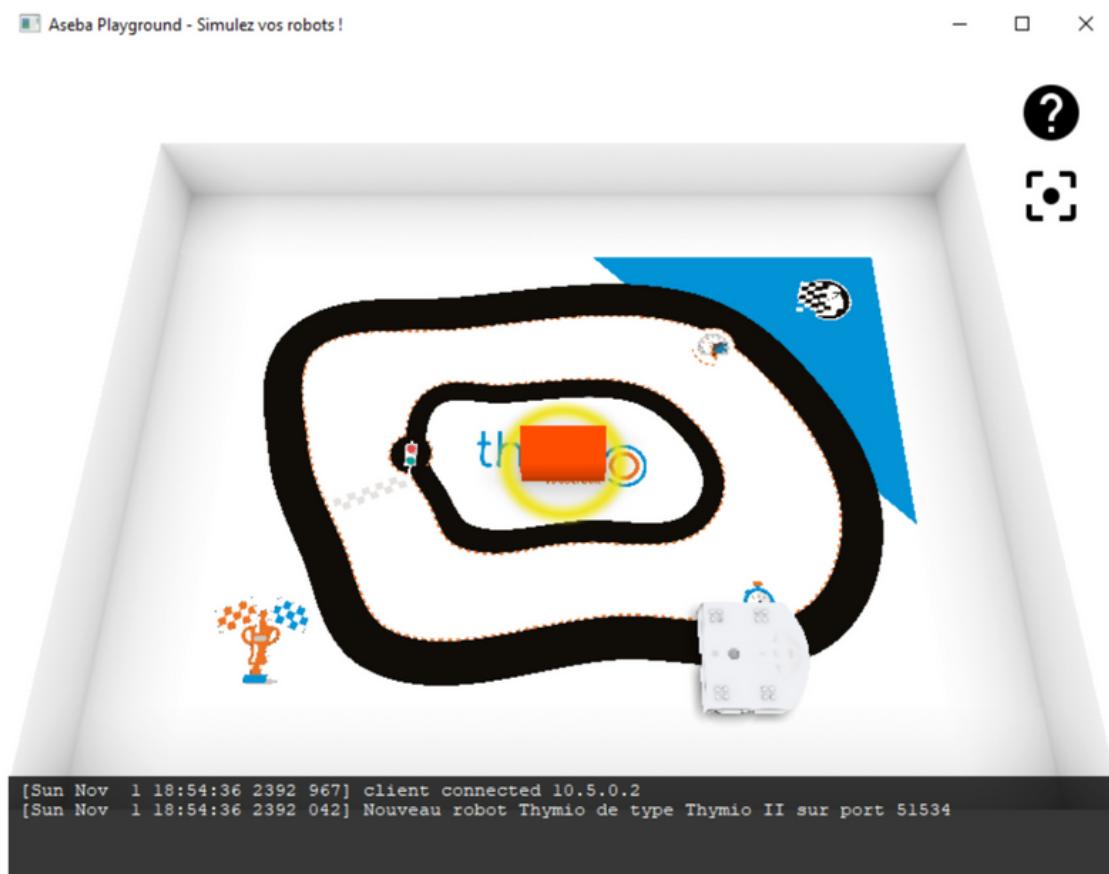
Le simulateur



Thymio Suite propose un simulateur 3D accessible par l'icône de la fenêtre d'accueil.
<https://www.thymio.org/fr/ressources/simulateur/>

Si l'on charge l'exemple nommé *thymio-challenge-pack.playground*, la fenêtre suivante apparaît.

<https://www.thymio.org/fr/ressources/ressource-dactivites/>



Cette scène 3D comporte plusieurs éléments

- Un cadre en creux dans lequel le robot peut évoluer
- Un robot Thymio
- Un pavé rectangulaire orange au centre
- Tout le reste est une image de fond.

La programmation s'effectue comme celle d'un robot physique avec quelques restrictions sur les capteurs et actionneurs. Les éléments disponibles figurent dans le tableau donné en Annexe.



Auteur: JOËL RIVET

Chapitre 3 - Thymio Suite et Thonny



Les interactions que l'on peut avoir avec la souris sont résumées ci-dessous :

- clic gauche :
 - sur un objet : le sélectionne. Si l'événement tap est actif, provoque un choc
 - sur le fond : désélectionne l'objet si il y a lieu.
- clic gauche + glisser :
 - sur le fond : déplace la vue (caméra)
 - sur un objet ou le robot : le déplace
- double clic gauche :
 - sur le robot ou un objet : zoom et passe en vue subjective (première personne)
 - sur le fond : aucun effet
- clic droit : aucun effet
- clic droit + glisser :
 - sur le fond : tourne la vue (caméra)
 - sur un objet : le tourne
- roue de la molette centrale : zoom
- appui molette centrale : aucun effet
- touche maj + clic + glisser: zoom comme la molette

Remarque : un objet n'est pas toujours déplaçable, tout dépend comme la scène a été construite.

Un éditeur de scène (ou terrain)

Aucun outil n'a été pour l'instant prévu pour modifier la composition des scènes en ajoutant des objet par exemple ou en changeant l'image de fond. C'est néanmoins possible. Le fichier playground est en fait une archive zip composé de fichiers que l'on peut modifier

Un éditeur de scène est en cours d'élaboration et sera très prochainement disponible.



Auteur: JOËL RIVET

Chapitre 3 - Thymio Suite et Thonny



Annexe : capteurs et actionneurs compatibles simulateur

Actionneur	Simulateur
Moteur	Oui
LEDs RVB dessus	Oui
LEDs RVB dessous	Non
LEDs cercle	Non
LEDs capteurs	Non
Audio	Non
Capteurs	
de proximité horizontal	Oui
Sol	Oui
Bouton	Oui
Accéléromètre	Oui*
Choc	Oui
Microphone	Non
Température	Non

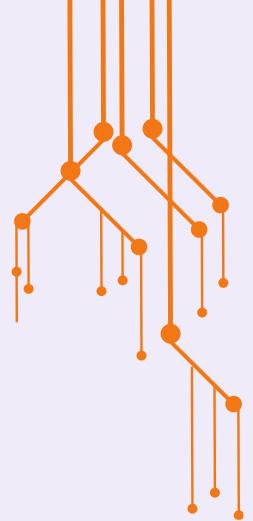
* sans effet car on ne peut pas pencher le robot du simulateur



Auteur: JOËL RIVET

Chapitre 3 - Thymio Suite et Thonny



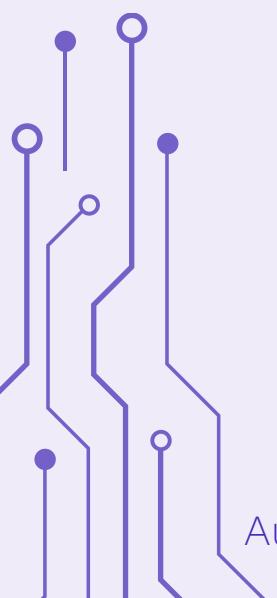


SESSION 1

CHAPITRE 4

Programmer capteurs
et actionneurs - 1

P18 - P29



Auteur: JOËL RIVET

CONTENTS

01

STRUCTURE D'UN
PROGRAMME PYTHON

02

CAPTEURS DE PROXIMITÉ ET
LES MOTEURS : LE PIÉTON

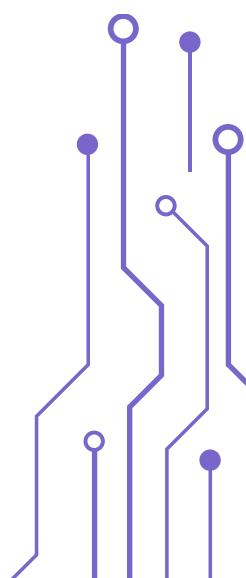
03

DES BOUTONS ET DES LEDS

ANNEXE 1 : EXERCICE 4.2 -
ÉVITEMENT - SOLUTION

ANNEXE 2 : EXERCICE 4.3 -
32000 COULEURS - SOLUTION

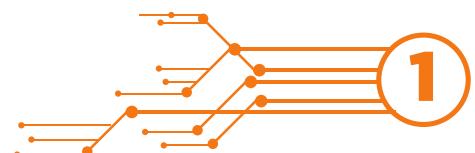
ANNEXE 3 : CODAGE RVB
DES LEDS



Il s'agit dans ce chapitre de programmer python pour prendre en charge les principaux capteurs et actionneurs de Thymio. On s'appuiera sur des scripts simples. Dans la suite nous utiliserons parfois le terme de script à la place de programme.

Afin de suivre avec profit ce chapitre, il est souhaitable que le lecteur dispose d'un Thymio connecté et de Thymio Suite pour appliquer les scripts proposés par la suite (voir chapitre 3).

Structure d'un programme python



Comportement général d'un robot

Nous avons vu au chapitre précédent que le rôle principal d'un programme de robotique est de répondre aux informations des capteurs par des actions appropriées, tout en poursuivant un but bien défini par le programmeur.

Par exemple, un robot doit se rendre d'un point A à un point B. Si en cours de route, il rencontre un obstacle, il doit l'éviter pour atteindre malgré tout le point B.

Ce comportement réactif du robot peut se décrire par des phrases comme :

Si un obstacle se trouve devant, **alors** le robot tourne à gauche ...

Structure d'un programme

Il comporte plusieurs parties :

- l'initialisation dans laquelle on crée des variables utilisateurs et fixe les valeurs des variables système (les variables attachées au robot).
- des blocs qui commencent par le décorateur **@onevent**, suivi de la définition d'une fonction qui gère les réponses aux événements reçus par la fonction. Il y a un bloc par famille de capteurs.
- éventuellement des fonctions définies par l'utilisateur.

L'ordre dans lequel ces parties sont disposées dans le code n'a pas d'influence dans son comportement. N'oublions pas que nous sommes dans le contexte d'une programmation événementielle et que les blocs **@onevent** sont parcourus en boucle plusieurs fois par seconde. En revanche, l'initialisation et les fonctions utilisateurs seront mémorisés à la première lecture du code.

Pour découvrir les capteurs de proximité et les moteurs et mettre en œuvre le comportement précédent, on propose le programme **Piéton**.

Limitation

Quand on élabore un programme python pour Thymio, il faut garder à l'esprit que ce code sera converti en Aseba, et doit donc respecter les contraintes de ce langage.

Pour autant, il n'est pas nécessaire d'apprendre le langage Aseba, les restrictions seront précisées au fur et à mesure.

Dans la session 2 avancée de ce cours, toutes ses particularités seront abordées.

Par exemple, le seul type autorisé est l'entier 16 bits signé. Donc on ne peut utiliser que les nombres entiers compris entre -32768 et 32767.

Capteurs de proximité et les moteurs : le piéton



On se propose de créer ce petit script qui évoque le comportement d'un véhicule autonome :

Si il n'y a pas d'obstacle devant le robot, il avance. Sinon, il s'arrête. On peut imaginer que l'obstacle est un piéton qui passe devant le robot.

Ce script utilise le capteur de proximité avant central de Thymio ainsi que les moteurs des 2 roues.

Avant de détailler les propriétés des capteurs et moteurs, réalisons le script en python.

La première des choses à faire est de définir une variable pour la vitesse. Elle peut aller de 0 à 500.

```
speed = 100 # ne pas mettre de valeur non entière
```

Puis une fonction liée aux capteurs de proximité

```
@onevent  
def prox():
```



Auteur: JOËL RIVET

Chapitre 4 - Programmer capteurs et actionneurs - 1



Fonctionnement des capteurs de proximité

Chaque capteur émet un rayonnement infra-rouge.

Si un obstacle se trouve devant, une partie du rayonnement revient dans le capteur, qui détecte alors un signal qui peut monter jusqu'à 5000 (unité arbitraire).

Si il n'y a rien devant, le rayonnement ne revient pas et le signal vaut alors 0.

Ce signal est stocké dans un tableau de 7 éléments nommé en python **prox_horizontal**.

La figure montre que par exemple, le capteur avant central correspond à la variable `prox_horizontal[2]`,

Le capteur arrière droit à la variable `prox_horizontal[6]`.

La limite de détection est d'environ 11 cm.



On traite le cas où le capteur central ne détecte rien, avec un seuil de détection fixé arbitrairement à 3500 (vous pouvez changer ce seuil pour tester). Le robot avance la vitesse `speed`.

```
if prox_horizontal[2] < 3500:  
    motor_left_target = speed  
    motor_right_target = speed
```

la variable **motor_left_target** fixe la vitesse de rotation du moteur gauche, **motor_right_target** celle du moteur droit.

puis on traite le cas où un obstacle est détecté (valeurs du capteur ≥ 3500).

```
else:  
    motor_left_target = 0  
    motor_right_target = 0
```

La fréquence d'appel à la fonction **prox**, donc la fréquence à laquelle les vitesses sont ajustées est de 10 Hz, soit toutes les 100 ms.

Dernière étape :

On recense dans la fonction les variables qui sont en écriture (`var = ...`) et on les déclare global au début :

```
global motor_left_target, motor_right_target
```

Fonctionnement des moteurs et des roues

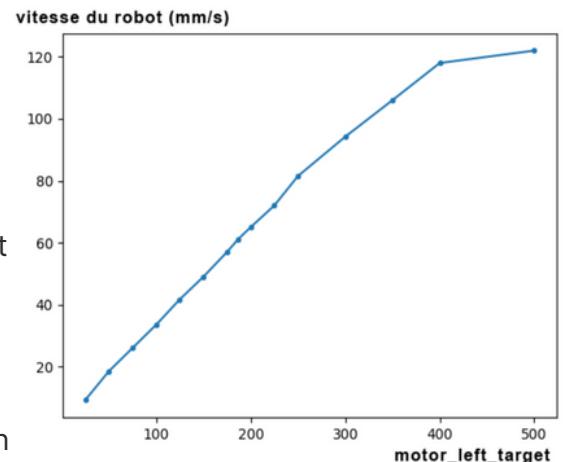
Les variables **motor_left_target** et **motor_right_target**

prennent leur valeur dans l'intervalle [0, 500].

target représente la consigne passée au moteur.

Il existe une autre variable **motor_left_speed** qui prend la valeur de la vitesse réelle mesurée du moteur. Cette valeur instantanée n'est jamais égale à celle de target et présente de grandes variations mais en moyenne la consigne target est assez bien respectée.

la courbe ci-contre donne la vitesse réelle du robot en fonction de la vitesse cible. Elle peut différer en fonction de la nature du sol et de l'état du robot.



Le code complet :

```
speed = 100

@onevent
def prox():
    global motor_left_target, motor_right_target

    if prox_horizontal[2] < 3500:
        motor_left_target = speed
        motor_right_target = speed
    else:
        motor_left_target = 0
        motor_right_target = 0
```

Exercice 4.2

Pour vous entraîner et vérifier vos acquis, un exercice vous est proposé.

Reprendre le programme précédent en modifiant le comportement. Si un objet se trouve devant, Thymio recule. Si un objet se trouve plutôt d'un côté, Thymio l'évite en tournant de l'autre. Si un obstacle est capté par un des 2 capteurs arrière, il avance. Dans le cas où il ne capte rien, il reste à l'arrêt.

Pour simplifier, on n'envisagera pas les cas où plusieurs de ces situations se présentent simultanément. Par exemple, s'il y a à la fois un objet devant et un derrière.

Solution en Annexe 1



Auteur: JOËL RIVET

Chapitre 4 - Programmer capteurs et actionneurs - 1

Des boutons et des LEDs

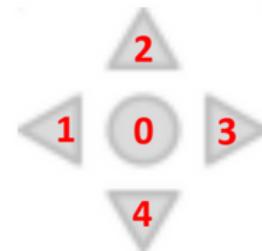


3.1 Les boutons

Thymio dispose de 5 boutons à l'avant en forme de flèche ou de cercle.

A chaque bouton correspond une variable système prédéfinie.

- bouton central (0 sur la figure) : **button_center**
- bouton de gauche (1) : **button_left**
- bouton avant (2) : **button_forward**
- bouton droit (3) : **button_right**
- bouton arrière (4) : **button_backward**



Chaque variable peut prendre 2 valeurs:

- la valeur 1 quand un doigt appuie sur un bouton
- la valeur 0 dans le cas contraire.

Par ailleurs, le bloc événement buttons est scruté 20 fois par seconde

```
@onevent  
def buttons():
```

Pour tester l'appui sur le bouton central par exemple, on utilisera le test :

```
if button_center == 1 :
```

3.2 Les LEDs

Thymio possède :

- 3 LEDS rvb (1 dessus, 2 dessous) dont la couleur est programmable (pour le codage rvb, voir Annexe 3).
- 1 cercle de 8 LEDs jaunes autour des boutons.
- d'autres LEDs liées aux capteurs et actionneurs, moins utiles pour le programmeur (pour plus de détail, voir le document API Thymio-python).

3.3 Programmation des LEDs : utiliser des variables système

Soit le script :

```
@onevent
def buttons():
    global leds_top

    if button_left == 1:
        leds_top = RED
```

la fonction **buttons()** étant parcourue 20 fois par seconde, si on appuie sur le bouton gauche au bon moment,

if button_left == 1: est évalué à vrai et les LEDs du dessus s'allume en rouge.

Il est facile de compléter le script pour que :

Si on appuie sur le bouton avant, les LEDs du dessus s'allument en rouge en vert.

Si on appuie sur le bouton droit, les LEDs du dessus s'allument en rouge en bleu.

Si on appuie sur le bouton central, les 3 couleurs sont activées, sur le bouton arrière, les LEDs s'éteignent

Le script complet :

```
@onevent
def buttons():
    global leds_top

    if button_left == 1:
        leds_top = RED
    elif button_forward == 1:
        leds_top = GREEN
    elif button_right == 1:
        leds_top = BLUE
    elif button_center == 1:
        leds_top = WHITE
    elif button_backward == 1:
        leds_top = BLACK
```



On observe que l'affichage des couleurs s'effectue par modification de valeurs d'une variable `leds_top`, comme dans le cas des moteurs.

Mais comment procéder si on veut utiliser d'autres couleurs ?

Il faut alors affecter à la variable `leds_top`, non une constante système comme `RED` mais une liste de 3 entiers.

Chaque entier compris entre 0 et 32, représente une composante rouge, verte, bleu d'une couleur.

Par exemple, si on désire produire la lumière orange, on écrira `leds_top = [32, 16, 0]`.

Ainsi, il existe 8 constantes qui désignent des couleurs :

nom	<code>RED</code>	<code>GREEN</code>	<code>BLUE</code>	<code>CYAN</code>	<code>MAGENTA</code>	<code>YELLOW</code>	<code>WHITE</code>	<code>BLACK</code>
composition	[32, 0, 0]	[0, 32, 0]	[0, 0, 32]	[0, 32, 32]	[32, 0, 32]	[32, 32, 0]	[32, 32, 32]	[0, 0, 0]

Les autres couleurs seront écrites avec une liste de 3 entiers.

Imaginons que nous voulions améliorer notre script pour afficher n'importe quelle nuance de couleur.

Nous avons à notre disposition $32 * 32 * 32 = 2^{15} = 32768$ combinaisons possibles.

Une solution consisterait à augmenter de 1 la valeur d'une composante quand on appuie sur le bouton correspondant.

Par exemple, mettons que la composante verte vale 18; en appuyant sur le bouton avant, elle passerait à 19. Quand on atteint 32, on peut décider qu'elle repasse à zéro.

D'un point de vue programmation, ça impose d'utiliser une variable pour la composante verte, `v` par exemple. On pourrait alors écrire :

```
if button_left == 1:  
    v = v + 1  
    leds_top = [r, v, b]
```

Hélas, ce code ne fonctionnera pas car il n'est pas possible d'écrire une liste contenant autres choses que des littéraux entiers ou des expressions produisant directement des entiers.

Ecrire `leds_top = [32, v, 0]` est interdit, `leds_top = [32, 18, 0]` est autorisé.

La raison à cela ? N'oublions pas que notre code python doit être traduit en Aseba et doit donc respecter les limitations propres à Aseba, qui n'accepte que des tableaux d'entiers (16 bits signés pour préciser).

Sommes-nous alors démunis de solution ? Non, il existe un autre moyen



3.4 Programmation des LEDs : utiliser une fonction

On peut également modifier une couleur avec la fonction **nf_leds_top(r, v, b)** qui prend en paramètres 3 entiers r, v , b représentant une composante rouge, verte ou bleu. Ils varient entre 0 et 32 inclus (annexe 3).

Par exemple **nf_leds_top(32, 0, 0)** est équivalent à **leds_top = RED**.

La différence avec les listes, c'est que les paramètres (ou arguments) de la fonction peuvent être des variables, car les fonctions acceptent les variables.

Donc pour augmenter la composante verte, on peut très bien écrire quelque chose comme :

v = 18

nf_leds_top(v, 3, 8)

v += 1

nf_leds_top(v, 3, 8)

En revanche, on ne peut pas utiliser les constantes prédéfinies ; on ne peut pas écrire :

nf_leds_top(YELLOW) car la fonction doit posséder 3 arguments exactement et YELLOW correspond à 1 seul argument.

Pour tester, dans le programme précédent, remplacer **leds_top = RED** par **nf_leds_top(32, 0, 0)**

3.5 Il y a bouton et bouton

Pour prendre en compte qu'un bouton a été appuyé, il y a 2 méthodes:

Méthode 1 :

Celle qui a été pratiquée précédemment. Au sein d'une fonction **buttons()**, on place le test **button_left == 1** (idem pour les autres boutons).

Méthode 2:

On peut aussi utiliser une fonction attachée à un seul bouton, par exemple :

def button_left(): (précédée du décorateur @onevent).

Dans ce cas, le code est :

```
@onevent
def button_left():
    global leds_top
    leds_top = RED
```



Auteur: JOËL RIVET

Chapitre 4 - Programmer capteurs et actionneurs - 1



Le tableau ci-dessous met bien en évidence la différence

Méthode 1

```
@onevent
def buttons():
    global leds_top

    if button_left == 1:
        leds_top = RED
```

Méthode 2

```
@onevent
def button_left():
    global leds_top
    leds_top = RED
```

Ces 2 méthodes sont-elles parfaitement équivalentes ? Pas tout à fait.

Dans le premier cas, la fonction **buttons()** est parcourue 20 fois par seconde. Ce qui signifie que si on laisse son doigt sur le bouton plus longtemps qu' 1/20 de s, ce qui est de loin le cas le plus fréquent, le test sera évalué plusieurs fois et son contenu exécuté plusieurs fois aussi, ce qui peut s'avérer très gênant si vous voulez par exemple incrémenter une variable.

Dans le deuxième cas, la fonction n'est parcourue qu'une fois (en fait 2 fois mais nous y reviendrons plus loin), ce qui produit une programmation plus naturelle.

C'est cette dernière méthode qu'il faudra utiliser dans l'exercice qui suit.

3.6 Exercice 4.3

Revenons à notre désir d'écrire un programme qui permette d'afficher les 32768 nuances de couleurs possibles.

Une approche consiste à partir du noir [0,0,0] et d'augmenter par pas de 1 la valeur d'une composante en appuyant sur un bouton, jusqu' 32. Avec 3 boutons, on couvre toutes les nuances.

Pour commencer, on initialise 3 variables à 0 :

```
r = 0
g = 0
b = 0
```

Assignons à chaque bouton un rôle :

En appuyant sur le bouton gauche, on augmente la composante rouge

En appuyant sur le bouton avant, on augmente la composante verte

En appuyant sur le bouton droit, on augmente la composante bleu

En appuyant sur le bouton arrière, on remet les composantes à 0.

Concentrons nous sur un bouton, le premier par exemple.

```
@onevent
def button_left(): # ne concerne que le bouton gauche
```



Nous indiquons la suite du code en français, laissant la charge au lecteur de traduire en python.

En appuyant sur le bouton gauche, on incrémente la variable r de 1.

si r dépasse 32, on ramène r à zéro

on allume la LED du dessus et pour bien suivre le programme dans la console de Thonny, on peut afficher les valeurs r, v, b avec **print(r, v, b)**

On fait de même pour les boutons avant et droit.

On finit avec le bouton arrière.

Et ne pas oublier de mettre au début de chaque fonction un clause **global** suivie des variables modifiées dans les fonctions.

Remarque : Si vous observez attentivement les print(), vous devriez observer que l'événement se déclenche deux fois pour un seul appui. Effectivement, 2 événements sont déclenchés, un quand le doigt touche le bouton et un deuxième quand le doigt est retiré. Pour les plus avancés de nos lecteurs, on peut ajouter un peu de code pour ne pas tenir compte du 2ième événement et incrémenter de 1 en 1 au lieu de 2 en 2, mais on vous laisse chercher ...

Solution en annexe 2.

Annexe : Exercice 4.2 - évitement - solution

```
speed = 100

@onevent
def prox():
    global motor_left_target, motor_right_target

    if prox_horizontal[2] > 3500:
        motor_left_target = -speed
        motor_right_target = -speed
    elif prox_horizontal[0] > 3500:
        motor_left_target = speed
        motor_right_target = 0
    elif prox_horizontal[4] > 3500:
        motor_left_target = 0
        motor_right_target = speed
    elif prox_horizontal[5] > 3500 or prox_horizontal[6] > 3500:
        motor_left_target = speed
        motor_right_target = speed
    else:
        motor_left_target = 0
        motor_right_target = 0
```



Annexe : Exercice 4.3 - 32000 couleurs - solution

```
r = 0
g = 0 # variable en anglais donc g pour green et non pas v pour
vert
b = 0

@onevent
def button_left():
    global leds_top, r
    r += 1
    if r > 32 : r = 0
    nf_leds_top(r, g, b)
    print(r, g, b)

@onevent
def button_forward():
    global leds_top, g
    g += 1
    if g > 32 : g = 0
    nf_leds_top(r, g, b)
    print(r, g, b)

@onevent
def button_right():
    global leds_top, b
    b += 1
    if b > 32 : b = 0
    nf_leds_top(r, g, b)
    print(r, g, b)

@onevent
def button_backward():
    global leds_top, r, g, b
    r = 0
    g = 0
    b = 0
    nf_leds_top(r, g, b)
```

Pour éviter de déclencher l'événement à la sortie du doigt, il faut ajouter au début des fonctions un test genre :

if button_left == 1: return

ou mettre le code dans le bloc **if button_left == 0.**



Auteur: JOËL RIVET

Chapitre 4 Programmer capteurs et actionneurs - 1



Annexe : Codage rvb* des LEDs

Un peu de physique de la lumière

Une source de lumière primaire est une source qui produit sa propre lumière. On trouve des exemples dans la nature comme le soleil ou le feu, ou dans le monde technologique comme les lampes ou les écrans d'ordinateurs.

La lumière naturelle blanche est composée en fait d'une myriade de radiations de couleurs très diverses.

Mais on s'est aperçu que l'on pouvait créer une couleur blanche en ajoutant seulement 3 couleurs : le rouge, le vert et le bleu (couleurs dites primaires). C'est suivant ce principe que les LEDs colorées, dites LED rvb fonctionnent.

En dosant finement les intensités de chacune de ces 3 couleurs on peut obtenir une variété de nuances extrêmement grande. Avec un écran classique numérique, on propose 256 nuances de rouge, 256 nuances de vert et 256 nuances de bleu ce qui porte le nombre de couleurs à $256^3 = 16\ 777\ 216$.

Chaque nuance est fixée par un nombre entier pris dans l'intervalle [0, 255]

Le tableau ci-dessous présente quelques couleurs courantes.

nom	blanc	noir	rouge	vert	bleu	cyan	magenta	jaune	orange	bleu ciel	gris clair	gris foncé	marron	vert olive
couleur														
r	255	0	255	0	0	0	255	255	255	135	200	80	139	107
v	255	0	0	255	0	255	0	255	128	206	200	80	69	142
b	255	0	0	0	255	255	255	0	0	250	200	80	18	35
statut			primaire	primaire	primaire	secondaire	secondaire**	secondaire						

Annexe : Codage rvb* des LEDs

* le sigle de codage s'écrit parfois **rvb** en français pour rouge, vert, bleu ou **rgb** en anglais pour red, green, blue.

** une couleur secondaire est une couleur complémentaire d'une couleur primaire, c'est-à-dire que l'addition des couleurs donne du blanc.

Par exemple, le magenta est la couleur complémentaire du vert car :

$$\text{vert} (0,255,0) + \text{magenta} (255,0,255) = \text{blanc}(255,255,255)$$

De même, toute couleur a une couleur complémentaire : le marron du tableau (139, 69, 18) a pour complémentaire la couleur définie par le triplet (117, 189, 238) qui est un joli bleu clair. En effet :

$$139 + 117 = 256$$

$$69 + 187 = 256$$

$$18 + 238 = 256$$



Auteur: JOËL RIVET

Chapitre 4 Programmer capteurs et actionneurs - 1

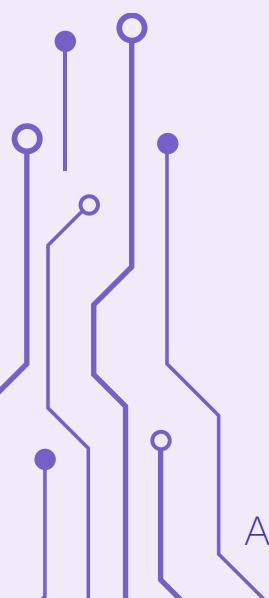


SESSION 1

CHAPITRE 5

Programmer capteurs
et actionneurs - 2

P30 - P45



Auteur: JOËL RIVET

CONTENTS

01

DES MINUTEURS ET DES SONS

02

ACCÉLÉROMÈTRE ET CHOCS

03

ÉVÉNEMENTIEL & SÉQUENTIEL

04

CAPTEURS SOL ET EXERCICE DE SYNTHÈSE

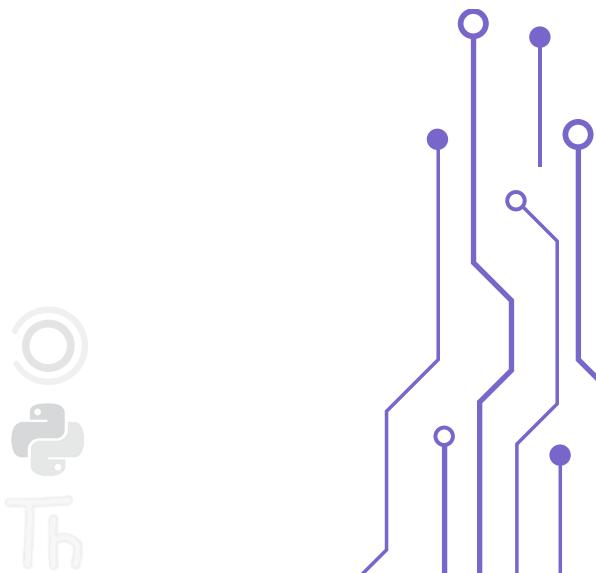
ANNEXE 1 - EXERCICE 5.1 -
POMPIERS - SOLUTION

ANNEXE 2 - EXERCICE 5.2 -
MODE AVION - SOLUTION

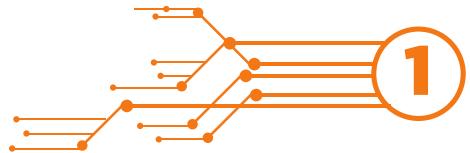
ANNEXE 3 - EXERCICE 5.2B -
DÉMARRAGE - SOLUTION

ANNEXE 4 - EXERCICE 5.3 - EXERCICE
DE SYNTHÈSE - SOLUTION

ANNEXE 5 - CORRESPONDANCE
NOTES - FRÉQUENCES



Des minuteurs et des sons



1.1 Les sons de Thymio

Thymio a plusieurs moyens de produire des sons:

- Comme pour les couleurs, il y a des sons système au nombre de 8.
 - Il y a aussi un générateur d'onde de synthèse qui peut jouer un son musical pour une durée donnée.
 - Le système peut aussi lire des sons enregistrés sur une carte micro SD logeable à l'arrière du Thymio.
- On peut même enregistrer de l'audio avec le micro et le rediffuser.

La qualité de l'ensemble reste modeste eu égard au microprocesseur et au système audio embarqué.

Les sons systèmes

Ils sont lues par la fonction **sound_system(n)** où n varie de -1 à 7. Au delà, le robot émet un son bref.

n peut prendre les valeurs (nomenclature aseba):

- 1 : arrête de jouer un son
- 0 : son de démarrage
- 1 : son d'arrêt (son non reconfiguration)
- 2 : son des boutons fléchés
- 3 : son du bouton central
- 4 : son de chute libre (peureux)
- 5 : son de choc
- 6 : son de suivi d'objet
- 7 : son de détection d'objet pour suivi

Les sons systèmes

Le générateur peut jouer un son continu de fréquence et de durée variable.

La fréquence peut varier de 55 Hz environ à quelques milliers de Hz. Mais l'onde qui sert de source au générateur est

très courte, ce qui fait que dans les aigus, le repliement de fréquence altère profondément le son.

La durée est exprimée en 1/60 de seconde.



Auteur: JOËL RIVET

Chapitre 5 - Programmer capteurs et actionneurs - 2



Exemple : produire la note la3 pendant 1/2 s s'écrit :

`nf_sound_freq(440, 30)` : 440 Hz est la fréquence de la note la3, et 30 donne 30/60 soit 0,5 seconde.

(pour une correspondance note-fréquence, voir Annexe 5).

Si on doit interrompre le son, utiliser -1 comme durée, peut importe la fréquence.

Il y a moyen de modifier la forme d'onde, ce sera abordé dans la session 2 de ce cours

Dans Thonny, pour un test rapide, créez un nouveau script. 

puis tapez : **nf_sound_freq(440, 30)** puis bouton 

L'utilisation de la carte SD

Ces possibilités seront abordées également dans la session 2 de ce cours.

1.2 Les minuteurs

Un paramètre que n'avons pas encore abordé, très important dans la programmation événementielle est le temps.

Pour gérer le temps, nous avons 2 outils à notre disposition :

- 2 minuteurs appelés timer0 et timer1
- une horloge, disponible en important le module clock. (voir session 2)

Un minuteur ou timer est accessible à travers :

- une variable (élément d'un tableau) : **timer_period[0]** ou **timer_period[1]**.
- une fonction liée à un événement : **timer0()** ou **timer1()**.

En voici le fonctionnement :

On commence par définir la durée du compte à rebours du minuteur, c'est-à-dire le temps au bout duquel le temps sera arrivé à zéro.

Soit par exemple **timer_period[0] = 2000 # en ms**. Cette affectation peut être placée n'importe où dans le code.

A partir du moment où le code lit cette instruction, le compte à rebours commence, indépendamment du reste du code qui poursuit son exécution de son côté. On parle de mode asynchrone.

Quand le compte à rebours se termine (ici au bout de 2000 ms soit 2s), le code va déclencher un événement et exécuter la fonction `timer0()` (qui devra être précédée par un décorateur). Il faut garder à l'esprit, qu'étant en mode asynchrone, cet appel de fonction interrompra toute exécution du code, quel qu'elle soit.

```
@onevent
def timer0():
    code à exécuter
```



Attention, les timer sont cycliques (périodiques), c'est-à-dire qu'une fois la fonction timer0 exécutée, le compte à rebours recommence. Pour arrêter le minuteur, il faut affecter 0 à la variable timer_period[0] :

timer_period[0] = 0. On place en général cette ligne dans la fonction timer0.

Remarque : tout ce qui vient d'être dit est applicable au 2ième minuteur timer1.

Exemple assez basique:

```
leds_top = RED # au début le robot est rouge
timer_period[0] = 1500 # lance le minuteur, le robot reste rouge

(quelque part dans le système, la variable timer_period[0] diminue de 1 à
chaque ms)

@onevent
def timer0(): # cette fonction est exécutée au bout de 1500 ms.
    global timer_period, leds_top
    leds_top = GREEN # le robot se colore en vert
    timer_period[0] = 0 # arrête le minuteur
```

En résumé, le robot reste rouge pendant 1,5 s puis devient vert.

1.3 Exercice 5.1 : Les pompiers

Mettons à profit ces 2 nouvelles fonctionnalités pour transformer Thymio en camion de pompier avec son fameux "pin pon".

Le cahier des charge est simple : Thymio avance en émettant son signal d'alert

démarrer

```
motor_left_target = 250
motor_right_target = 250
```

lancer le minuteur sur 1/2 seconde

```
timer_period[0] = 500
```

alternance : pour le son, nous allons alterner deux sons grave et aigu. Pour cela, il nous faut une variable booléenne

nommée up par exemple, qui bascule entre les valeurs vrai et faux.

Si up est vrai, jouer le son aigu (fréquence 440Hz), si up est faux, jouer le son grave (294 Hz)

Le laisse au lecteur le soin d'écrire la fonction

```
@onevent
def timer0():
```

Solution en annexe 1.



Auteur: JOËL RIVET

Chapitre 5 - Programmer capteurs et actionneurs - 2



Accéléromètre et chocs



Thymio dispose d'un accéléromètre qui peut remplir 2 fonctions :

- Déterminer une inclinaison du robot avec l'événement **acc**
- Déetecter un choc avec l'événement **tap**

2.1 Détecter une inclinaison

Sans rentrer dans les détails, cet accéléromètre est capable de déterminer son inclinaison par rapport à la verticale, qui est la direction de la pesanteur terrestre. Il peut donc mesurer l'inclinaison du robot suivant un axe avant-arrière ou un axe gauche-droite. Étant donné que la gravitation est verticale, il ne peut pas mesurer une rotation dans un plan horizontal.

On dispose d'un tableau **acc** de 3 entiers qui varient de -32 à +32:

acc[0] : positif si inclinaison vers la gauche, négatif vers la droite

acc[1] : positif si inclinaison vers l'arrière, négatif vers l'avant

acc[2] : ne donne pas de données nouvelles et pertinentes

Remarque : les valeurs données par l'accéléromètre ne sont pas précises, elles sont intéressantes pour établir des comparaisons ou pour indiquer une tendance.

Voici un petit script d'application directe. Observer notamment le lien entre l'inclinaison et la hauteur du son.

```
@onevent
def acc():

    nf_sound_freq(440 - 10 * acc[0], 15)
    if acc[0] < 0:
        nf_leds_top(32, 16, 0)
    else:
        nf_leds_top(32, 0, 16)
```

Exercice 5.2

On se propose de réaliser un programme indique les inclinaisons de Thymio par la lumière et le son.

Plus Thymio penche vers l'avant, plus il devient rouge et plus le son devient aigu, et les 3 leds circulaire de devant s'allument.

Plus Thymio penche vers l'arrière, plus il devient vert et plus le son devient grave et les 3 leds circulaires arrière s'allument.

En vous inspirant de l'exemple du paragraphe, réaliser le programme demandé.

Solution en annexe 2



Auteur: JOËL RIVET

Chapitre 5 - Programmer capteurs et actionneurs - 2



2.1 Déetecter un choc

Un choc n'est rien de plus qu'une brutale variation d'accélération et donc l'accéléromètre y est sensible.

La fonction correspondante à cet événement est:

```
@onevent  
def tap():
```

Contrairement à un certain nombre d'autres fonctions événement, la fonction tap() n'est pas parcourue systématiquement mais seulement en cas de choc.

Exemple dans lequel une tape sur le robot l'allume en orange:

```
@onevent  
def tap():  
    nf_leds_top(32, 16, 0)
```

Exercice 5.2b

Ecrire un petit script qui fait démarrer le robot avec une petite tape. Le robot émet également le son système de démarrage et se colore en jaune.

Une nouvelle tape l'arrête en éteignant les LEDs et en émettant le son système d'arrêt du robot.

Solution annexe 3



Auteur: JOËL RIVET

Chapitre 5 - Programmer capteurs et actionneurs - 2

Événementiel & séquentiel



La problématique

Pour saisir la problématique, imaginons qu'un robot doit adopter le comportement suivant :

- il avance et s'arrête au premier obstacle qu'il rencontre. Quand l'obstacle est retiré, il avance à nouveau.
- puis il rencontre un ligne noire qu'il traverse. Et à partir de ce moment, on désire que son comportement change.

Dorénavant, s'il rencontre un obstacle, il devra cette fois-ci amorcer un virage à droite au lieu de s'arrêter.

On a là un comportement typiquement séquentiel :

séquence 1 - le robot attend devant un obstacle

séquence 2 - le robot tourne devant un obstacle et devient insensible à tout autre obstacle en ne tenant plus compte de ses capteurs.

Comment faire adopter au robot un tel comportement dans un programme événementiel ?

Dans un tel programme, n'oublions pas qu'une boucle unique parcours le programme en attente d'éventuels

événements. On ne peut pas déroger à cette règle. On ne voit pas dans le code un début et une fin, la boucle peut potentiellement tourner indéfiniment.

Une astuce consiste à utiliser des variables appelées drapeaux (flags) ou variables d'état.

Ces variables permettent

d'autoriser certaines actions et pas d'autres et de prendre en compte certains événements ou non.

Dans notre scénario précédent, on doit pouvoir dire.

On place le robot dans l'état 1. Dans cet état, il doit s'arrêter devant un obstacle.

On place le robot dans l'état 2. Dans cet état, il doit amorcer un virage et devenir 'aveugle' aux capteurs.

La ligne noire est juste là pour changer l'état du robot (on pourrait mettre un timer à la place ou tout autre événement).

Il suffit de créer une variable appelée state dont la valeur nous permettra de sélectionner les états.

Un changement de couleur permet de visualiser les différents états.



Voici le code pour notre exemple :

```
state = 1
leds_top = YELLOW
motor_left_target = 100
motor_right_target = 100
@onevent
def prox():
    global state, motor_left_target, motor_right_target

    if state == 1:
        if prox_horizontal[2] > 3500:
            motor_left_target = 0 # le robot s'arrête
            motor_right_target = 0
        else:
            motor_left_target = 100
            motor_right_target = 100
        if state == 2:
            if prox_horizontal[2] > 3500:
                motor_left_target = 100 # le robot tourne
                motor_right_target = -100

    if prox_ground_delta[0] < 400: # permet de basculer de l'état 1 au 2

state = 2
leds_top = MAGENTA
```

On peut faire ainsi adopter des comportements différents au sein d'une même boucle événementielle

Quelques activités dans la session 2 utiliseront parfois une dizaine d'états sans que cela crée de confusion particulière

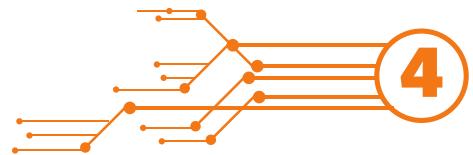


Auteur: JOËL RIVET

Chapitre 5 - Programmer capteurs et actionneurs - 2



Capteurs sol et exercice de synthèse



4

4.1 Capteurs sol

Thymio possède en plus de ses capteurs horizontaux 2 capteurs situés dessous à l'avant. On accède à leur mesures par 2 variables `prox_ground_delta[0]` (capteur gauche) et `prox_ground_delta[1]` (capteur droite). Ce sont les 2 éléments du tableau `prox_ground_delta`.

Leur domaine de valeurs est un peu plus restreint : [0, 1000]. Ils fonctionnent comme les autres par réflexion d'un rayon infra-rouge. Donc :

- Si le sol du dessous du robot est blanc, réflexion maximale, `prox_ground_delta[0 ou 1]` prend une valeur proche de 1000.
- Si le sol du dessous du robot est très noir ou absent (on tient le robot retourné en l'air), réflexion minimale ou nulle, `prox_ground_delta[0 ou 1]` prend une valeur proche de 0.

Application : suivre une ligne

On suppose que Thymio se trouve sur une ligne courbe noire de 4cm de largeur environ.

L'état des capteurs sol va

conditionner les actions à effectuer :

capteur gauche	capteur droit	situation	action
noir	noir	le robot est sur la ligne	tout droit
noir	blanc	le robot sort de la ligne par la droite	tourner légèrement à gauche
blanc	noir	le robot sort de la ligne par la gauche	tourner légèrement à droite
blanc	blanc	le robot est complètement sorti de la ligne	pivoter pour reprendre la ligne



Auteur: JOËL RIVET

Chapitre 5 - Programmer capteurs et actionneurs - 2



D'où le script

```
speed = 250
@onevent
def prox():
    global motor_left_target, motor_right_target

    if prox_ground_delta[0] < 400 and prox_ground_delta[1] < 400:
        motor_left_target = speed
        motor_right_target = speed

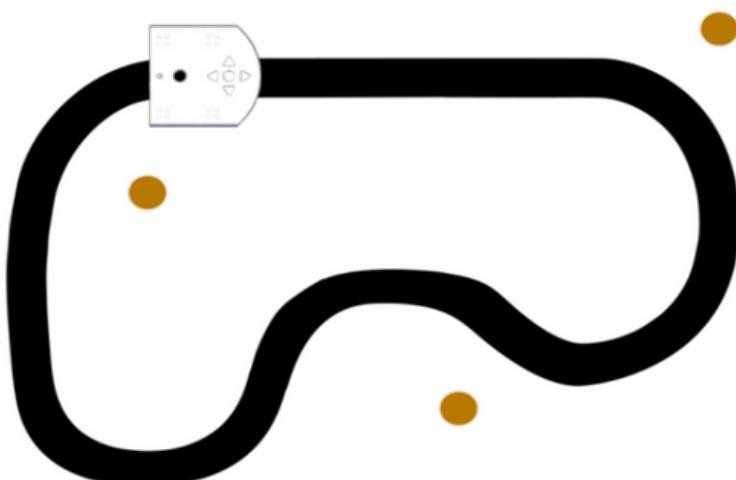
    elif prox_ground_delta[0] < 400 and prox_ground_delta[1] > 400:
        motor_left_target = -speed //5
        motor_right_target = speed

    elif prox_ground_delta[0] > 400 and prox_ground_delta[1] < 400:
        motor_left_target = speed
        motor_right_target = -speed //5

    elif prox_ground_delta[0] > 400 and prox_ground_delta[1] > 400:
        motor_left_target = speed
        motor_right_target = - speed
```

4.2 Exercice 5-3 : La boucle des lumières

Thymio se trouve au début d'une ligne noire bouclée qu'il doit suivre. Un appui sur le bouton avant permet de le faire démarrer. Chaque fois qu'il rencontre à gauche ou à droite un poteau (mettons 3 poteaux en tout) , il se colore parmi une des 8 couleurs [WHITE, RED, GREEN, BLUE, CYAN, MAGENTA, YELLOW, BLACK] choisie aléatoirement. Si un mur se trouve sur son chemin, il s'arrête et émet un son insistant tant que l'on ne retire pas l'obstacle. Une fois l'obstacle retiré, il poursuit sa route. On peut l'arrêter en appuyant sur le bouton arrière.



Aide pour le choix d'une couleur aléatoire:

Utiliser la fonction **nf_leds_top(r, v, b)**. r, v et b peuvent prendre seulement les 2 valeurs 0 ou 32. Ce qui fait 8 couleurs en comptant en plus blanc et noir.

On utilise pour les choix des 3 paramètres la fonction **math_rand()**.

Cette fonction (sans paramètre) fournit un entier compris dans l'intervalle [-32768, 32767].

Pour choisir par exemple si r doit valoir 0 ou 32, on peut regarder si math_rand() est pair ou impair avec l'opérateur modulo. Ainsi, si un entier n est pair, n modulo 2 vaut 0 et pour n impair modulo 2 vaut 1.

On multiplie le résultat par 32 pour avoir plutôt 0 ou 32 que 0 ou 1.

Ça donne finalement $r = 32 * (\text{math_rand()} \% 2)$ (en python, on rappelle que % est l'opérateur modulo)

Aide à la conception :

- Commencer par dessiner une ligne courbe fermée noire de 4 cm de largeur assez grande et surtout sans faire de virage trop serré.
- Puis faire un script où Thymio suit la ligne avec le tableau prox_ground_delta.
- Mettre en place le démarrage et l'arrêt avec le bouton central
- Une fois que ça marche, placer un petit obstacle sur un côté et changer la couleur et jouer un son
- Puis mettre 3 obstacles et programmer des couleurs aléatoires.
- Une fois que ça marche, programmer l'arrêt et un son grave si prox_horizontal[2] capte quelque chose

Solution en annexe 4



Auteur: JOËL RIVET

Chapitre 5 - Programmer capteurs et actionneurs - 2



Annexe 1 - Exercice 5.1 - pompiers solution

```
up = True
timer_period[0] = 500
motor_left_target = 250
motor_right_target = 250
@onevent
def timer0():
    global up

    if up:
        nf_sound_freq(440, 30)
    else:
        nf_sound_freq(394, 30)

    up = not up
```



Annexe 2- Exercice 5.2 - mode avion solution

```
inc = 0 # inc pour inclinaison

@onevent
def acc():
    global inc

    inc = acc[1]
    if inc < 0:
        nf_leds_top(inc, 0, 0)
        nf_sound_freq(440 - 10 * inc, 15)
        nf_leds_circle(32,32,0,0,0,0,0,32)
    else:
        nf_leds_top(0, inc, 0)
        nf_sound_freq(440 - 10 * inc, 15)
        nf_leds_circle(0,0,0,32,32,32,0,0)
```



Auteur: JOËL RIVET

Chapitre 5 - Programmer capteurs et actionneurs - 2



Annexe 3- Exercice 5.2b - démarrage solution

```
waiting = True

@onevent
def tap():
    global leds_top, waiting, motor_left_target, motor_right_target

    if waiting:
        nf_sound_system(0)
        leds_top = [32, 32, 0]
        motor_left_target = 300
        motor_right_target = 300
        waiting = False
    else:
        nf_sound_system(1)
        leds_top = [0, 0, 0]
        motor_left_target = 0
        motor_right_target = 0
        waiting = True
```



Annexe 4 - Exercice 5.3 - Exercice de synthèse - solution

```
speed = 250
r = 0
v = 0
b = 0
nf_leds_top(0,0,0)
run = False

@onevent
def button_forward():
    global run
    run = True # autorisation de rouler

@onevent
def button_backward():
    global run
    run = False

@onevent
def prox():
    global motor_left_target, motor_right_target, r,v,b, run

    if run: # autorisation de rouler
        # partie rester sur la ligne
        if prox_ground_delta[0] < 400 and prox_ground_delta[1] < 400:
            motor_left_target = speed
            motor_right_target = speed

        elif prox_ground_delta[0] < 400 and prox_ground_delta[1] > 400:
            motor_left_target = -speed //5
            motor_right_target = speed

        elif prox_ground_delta[0] > 400 and prox_ground_delta[1] < 400:
            motor_left_target = speed
            motor_right_target = -speed //5

        elif prox_ground_delta[0] > 400 and prox_ground_delta[1] > 400:
            motor_left_target = 0
            motor_right_target = 0

    # partie gestion d'un obstacle sur la piste
    if prox_horizontal[2] > 4500:
        motor_left_target = 0
        motor_right_target = 0
```



```
# partie détection d'un poteau latéral
elif prox_horizontal[0] > 2000 or prox_horizontal[4] > 2000:
    a = math_rand() // 117
    b = math_rand() // a
    r = 32 * (abs(b) % 2)

    a = math_rand() // 117
    b = math_rand() // a
    v = 32 * (abs(b) % 2)

    a = math_rand() // 117
    b = math_rand() // a
    b = 32 * (abs(b) % 2)
    nf_leds_top(r, v, b)

else:
    run = False # ne plus rouler
    motor_left_target = 0
    motor_right_target = 0
    motor_left_target = 0
    motor_right_target = 0
```



Annexe 5 - Correspondance notes fréquences

Un peu de physique du son

Une note de musique est un son dont la hauteur peut être identifiée sans ambiguïté. A chaque note correspond une fréquence. La note de référence est le **la** avec un fréquence de 440 Hz.

Toutes les autres notes peuvent être calculées à partir de celle-ci car l'ensemble constitue une suite géométrique de raison $r = \sqrt[12]{2} = 1,059$ environ. C'est-à-dire que pour passer d'une note à la suivante, distante d'un demi-ton, il faut multiplier ou diviser par r.

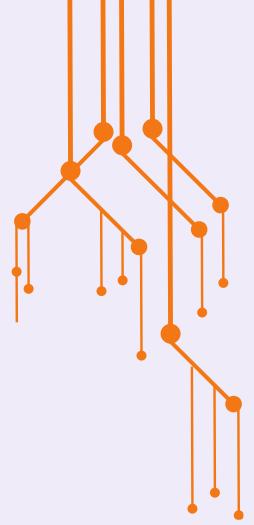
Ainsi, la note sib correspond à une fréquence de $440 \times \sqrt[12]{2} = 466$ Hz.

Ci dessous un tableau de quelques notes / fréquences (Hz):

sol2	lab2	la2	sib2	si2	do3	do#3	ré3	Mib3	mi3	fa3	fa#3	sol3
196,0	207,7	220,0	233,1	246,9	261,6	277,2	293,7	311,1	329,6	349,2	370,0	392,0

lab3	la3	sib3	si3	do4	do#4	ré4	Mib4	mi4	fa4	fa#4	sol4	lab4
415,3	440,0	466,2	493,9	523,3	554,4	587,3	622,3	659,3	698,5	740,0	784,0	830,6



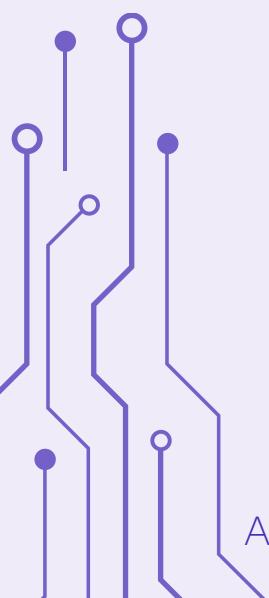


SESSION 1

ANNEXE

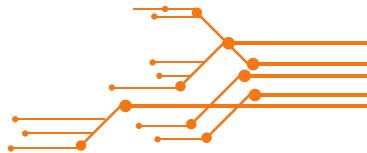
API Thymio Python

P46 - P54



Auteur: JOËL RIVET

API Python pour Thymio Résumé



Version Thymio Suite 2.4.0

Lorsqu'un événement se produit, la fonction correspondante (précédée d'un décorateur) est exécutée

Exemple :

```
@onevent
def buttons():
    code ....
```

Le même terme peut servir à désigner un événement (ex : button) et la fonction associée (ex : buttons())

Boutons (Buttons)

Fonctions

- **buttons()** : visité par défaut, que l'on appuie sur un bouton ou non. fréquence : 20 Hz.
- **button_forward(), button_right(), button_backward(), button_left(), button_center()** : visités à 20 Hz le temps d'appui sur le bouton-flèche correspondant, avant, droit, arrière, gauche, centre;

Variables

En lecture, écriture

- les variables suivantes sont à 1 le temps de l'appui d'un doigt seulement, sinon valeur à 0. Elles s'utilisent dans la fonction buttons().
- **button_backward** : flèche arrière
- **button_left** : flèche gauche
- **button_center** : bouton central
- **button_forward** : flèche avant
- **button_right** : flèche droite

Remarque : le mode écriture permet de simuler un appui sur le bouton.



Auteur: JOËL RIVET

Session 1 - Annexe - API Thymio Python



Capteurs de proximité (Prox)

Fonction

- prox() : visitée par défaut, fréquence : 10 Hz

Variables

En lecture, écriture

- **prox_horizontal[0-7]** : valeur des capteurs avant et arrière, (0 avant gauche, 1 avant centre-gauche, 2 avant centre, 3 avant centre droit, 4 avant droit, 5 arrière gauche, 6 arrière droit) : valeur de 0 à 4500 environ.
- **prox_ground_reflected** : quantité de lumière reçue pendant que le capteur émet de l'infrarouge, varie entre 0 (aucune réflexion) et 1023 (lumière maximale)
- prox_ground_delta : différence entre la lumière réfléchie et la lumière ambiante, liée à la distance et à la couleur du sol.

Remarque : la variable `asesba prox.ground.ambiant` n'est pas implémenté

En pratique, l'utilisation de **prox_ground_delta** est suffisante.

Communication (Prox_comm)

L'événement **prox_comm** se déclenche seulement en mode réception et si une donnée est reçue.

Fonctions

- **nf_prox_comm_enable(0 | 1)** : les capteurs horizontaux perdent leur fonction de capteur de détection de proximité d'un obstacle au profit de la communication (émission ou réception) d'un signal infra-rouge, émis ou reçu par un autre robot.
- 0 : mode détection (par défaut) , 1: mode communication
- **prox_comm()** : on placera la variable de réception dans cette fonction.

Variables

En émission

- **prox_comm_tx = n**: le robot émet la valeur *n* avec une fréquence de 10 Hz . Valeur entre -512 et 511. Peut se placer n'importe où dans le code.

En réception

- *n* = **prox_comm_rx** : le robot reçoit la valeur *n* à 10 Hz



Auteur: JOËL RIVET

Session 1 - Annexe - API Thymio Python



Accéléromètre (acc et tap)

Fonctions

- **acc()** : visitée par défaut, fréquence : 16 Hz
- **tap()** : se déclenche lors d'un choc

Variables

En lecture, écriture

- **acc[0]** : axe x (de droite à gauche du robot, positif si vers gauche)
- **acc[1]** : axe y (d'avant vers l'arrière du robot, positif si vers arrière)
- **acc[2]** : axe z (qui rentre du plan horizontal, positif si vers sol)

Les valeurs dans ce tableau varient de -32 à 32. L'accélération de la pesanteur (1 g = 9,81 m.s⁻²) correspond à 23, et donc une unité correspond à 0,40 m.s⁻². Thymio met à jour ce tableau à une fréquence de 16 Hz, et génère l'événement **acc** après chaque mise à jour. En outre, quand un choc est détecté, un événement **tap** est émis.

Température (temperature)

Fonction

- **temperature()** : visitée par défaut, fréquence : 1 Hz

Variable

En lecture, écriture

- **temperature** : valeur de la température en ° centigrade à 0,1 près. Thymio met à jour cette valeur à 1 Hz, et génère l'événement **temperature** après chaque mise à jour.

Remarque : la valeur de la température est au-dessus (env 3°) de la température ambiante car le capteur est sensible à la température de fonctionnement du robot.



Auteur: JOËL RIVET

Session 1 - Annexe - API Thymio Python



LEDs

Les leds (lumières) sont des actionneurs qui ne déclenchent aucun événement.

On peut contrôler les actionneurs en modifiant les valeurs de variables ou en appelant les fonctions dédiées.

Toutes les valeurs sont comprises entre 0 (éteint) et 32 (éclairement maximal)

Fonctions

Leds rvb

- **nf_leds_top(red, green, blue)** : situés dessus le robot, commande les valeurs de rouge, vert et bleu respectivement.
- **nf_leds_bottom_left(red, green, blue)** : leds du dessous à gauche
- **nf_leds_bottom_right(red, green, blue)** : leds du dessous à droite

Leds jaunes en cercle

- **nf_leds_circle(led0, led 1, led 2, led 3, led 4, led 5, led 6, led 7)** où led n commande l'intensité de la LED avant du robot (numérotation dans le sens des aiguilles d'une horloge, led0 correspond à midi, au-dessus de la flèche avant).

Leds des capteurs de proximité

Chacun des capteurs de proximité du robot est accompagné d'une LED rouge soudée juste à côté (deux pour le capteur du milieu avant).

Activation par défaut : s'allument en présence d'un objet devant le capteur associé, avec une intensité plus forte plus l'objet est proche.

- **nf_leds_prox_h(led0, led 1, led 2, led 3, led 4, led 5, led 6, led 7)** (voir capteur pour les indices)
- **nf_leds_prox_v (led 0, led 1)** commande les LEDs des capteurs dessous, gauche et droite.

Leds des boutons

4 LEDs rouges sont placées entre les boutons sur Thymio.

Activation par défaut : Pour chacun des boutons flèches, une LED s'allume lorsqu'il est appuyé. Pour le

bouton du centre, les 4 LEDs s'allument.

- **nf_leds_buttons(led 0, led 1, led 2, led 3)** commande ces LED avec le code 0 avant, 1 droite, 2 arrière, 3 gauche

Led du récepteur télécommande

Cette LED rouge est placée à côté du récepteur télécommande à droite.

Activation par défaut : clignote lorsqu'un code RC5 est reçu.

- **nf_leds_rc(led)** permet de la contrôler.



Auteur: JOËL RIVET

Session 1 - Annexe - API Thymio Python

Leds du capteur de température

Ces deux LEDs (une rouge, une bleue) se trouvent placée à côté du capteur de température.

Activation par défaut : rouge si la température est au dessus de 28°C, rouge et bleu entre 28° et 15°, bleu

pour une température en dessous de 15°.

- **nf_leds_temperature(red, blue)** permet de les contrôler.

Led du micro

Cette LED bleue est placée à côté du microphone.

Activation par défaut : éteinte.

- **nf_leds_sound(led)** permet de la contrôler.

Il y a aussi des LEDs qui ne peuvent pas être contrôlées par l'utilisateur :

- 3 LEDs vertes dessus indiquent le niveau de batterie ou de mise en charge.
- une LED bleue et une rouge derrière indiquent l'état de la charge.
- une LED rouge à l'arrière est liée à la carte SD.

Variables

Certaines fonctions précédentes peuvent être remplacées par des écritures dans les variables suivantes :

leds_top, leds_bottom_left, leds_bottom_right, leds_circle

exemple :

- **leds_top = [32, 0, 32]** # produit la couleur magenta
- **leds_circle[0] = 32**

Astuce : Les 8 valeurs du tableau leds_circle peuvent donc être utilisées comme des variables d'état lumineuses

Constantes

Pour les couleurs, Thymio possède 8 couleurs prédéfinies qui correspondent aux couleurs de base. Elles sont en majuscule, comme c'est souvent le cas pour des constantes.

WHITE, RED, GREEN, BLUE, CYAN, MAGNETA, YELLOW et BLACK (lumière éteinte)

Ces couleurs sont utilisables avec les variables, mais pas avec les fonctions.

L'affectation **leds_top = RED** est valide mais la fonction **nf_leds_top(RED)** ne marche pas. En effet la fonction **nf_leds_top** demande 3 paramètres . Or la constante **RED** cache les 3 paramètres dans une liste : **RED** est strictement identique à **[32, 0, 0]** ce qui ne fait qu'un paramètre.



Moteurs (motors)

Fonction

- **motor** : visité par défaut, fréquence : 100 Hz

Variables

En lecture, écriture

- **motor_left_target** : vitesse demandée roue gauche
- **motor_right_target** : vitesse demandée roue droite

En lecture

- **motor_left_speed** : vitesse réelle roue gauche
- **motor_right_speed** : vitesse réelle roue droite
- **motor_left_pwm** : valeur de commande roue gauche
- **motor_right_pwm** : valeur de commande roue droite

Les valeurs des variables de type target et speed sont comprises entre -500 et 500.

Les valeurs des variables de type pwm ne sont pas documentées

Détection de l'intensité du son (mic)

Thymio peut détecter lorsque le bruit ambiant est au-dessus d'une intensité donnée et émet alors un événement.

Fonction

- **mic()** : exécutée si *mic.intensity* est au-dessus de *mic.threshold*

Variables

En lecture

- **mic_intensity** : intensité du bruit capté par le microphone, les valeurs sont comprises entre 0 et 255

En écriture

- **mic_threshold** : seuil de détection du micro pour déclencher la fonction mic(). Valeurs entre 0 et 255.



Auteur: JOËL RIVET

Session 1 - Annexe - API Thymio Python



Télécommande (RC5)

Thymio possède un détecteur de télécommande infrarouge compatible RC5

Fonction

- **rc5()** : exécutée quand un signal de télécommande est reçu.

Variables

En lecture

- **rc5_address** : 0 pour tous les robots, donc pas de possibilité de distinguer les robots, tous les robots captent les mêmes commandes.
- **rc5_command** : un entier qui correspond à une des touches de la télécommande
 - bouton de gauche : 85
 - bouton de droite : 86
 - bouton central : 87
 - bouton supérieur : 80
 - bouton inférieur : 81

Minuterie (timer0, timer1)

Thymio fournit deux timers définis par l'utilisateur.

Événement

- **timer0()** : exécutée lorsque la variable timer_period [0] arrive à zéro
- **timer1()** : exécutée lorsque la variable timer_period [1] arrive à zéro

Variables

En écriture, lecture

Un tableau de 2 valeurs, timer.period , permet de spécifier la période des timers :

- **timer_period [0]** : période du timer0 en milliseconde
- **timer_period [1]** : période du timer1 en milliseconde

Lorsque le délai expire, le timer génère un événement timer0 (ou timer1).

On parle de *period* car une fois un temps écoulé, le décompte reprend automatiquement.

Mettre explicitement les variables *timer_period [0]* ou *timer_period [1]* à 0 si on veut arrêter les timer.



Auteur: JOËL RIVET

Session 1 - Annexe - API Thymio Python



Les sons

Thymio gère 3 types de sons

- Les sons système au nombre de 8
- Les sons de synthèse
- Les sons enregistrés sur une carte micro-SD.

La fin de la lecture des sons système ou de synthèse provoque l'événement **sound_finished**.

Fonction

nf_sound_finished(): est exécutée quand la lecture d'un son s'achève de lui-même. Si le son est interrompu avant, il ne se déclenche pas.

Sons système (fonctions)

nf_sound_system(val) : génère un son préenregistré. valeur comprise entre -1 et 7

Les sons suivants sont disponibles :

paramètre description

- 1 arrête de jouer un son
- 0 son de démarrage
- 1 son d'arrêt (son non reconfiguration)
- 2 son des boutons fléchés
- 3 son du bouton central
- 4 son de chute libre (peureux)
- 5 son de choc
- 6 son de suivi d'objet
- 7 son de détection d'objet pour suivi

Son de synthèse

- **nf_sound_freq(f, d)** : génère un son assimilable à une note de musique de fréquence *f* en Hz et de durée *d*.
d est exprimée en 1/60 de seconde. Par exemple si *d*=30, *d* = 30/60 = 0,5 s.
- **nf_sound_wave(tab)** : permet de définir sa forme d'onde. tab est une liste de 142 valeurs (échantillons), les valeurs sont comprises entre -128 et 127. Cette liste-forme d'onde devra être lue par la fonction nf_sound_freq().
- **nf_sound_duration(n)** : renvoie une valeur non documentée. Pour obtenir la durée réelle d'émission d'un son, utiliser la fonction nf_sound_finished() associée au module **clock**.

Sons SD

Il faut connecter une carte micro-SD à l'arrière du robot. Toutes les cartes ne fonctionnent pas. Il faut une carte au format SD standard, classes 2 ou 4, de faible capacité, difficile à trouver car elles sont anciennes. On trouve encore des classe 10 (SDHC) à tester.

(détail : <https://www.sdcard.org/developers/overview/capacity/index.html>)

Le site idealo.fr propose ces anciennes cartes.



Auteur: JOËL RIVET

Session 1 - Annexe - API Thymio Python



Enregistrement (fonction)

- **nf_sound_record(n)** : déclenche l'enregistrement du son capté par le micro de Thymio. Ce son est stocké sur la carte SD avec le nom *Rn.wav*. n est un entier compris entre 0 et 32737.

Pour arrêter l'enregistrement, utiliser la fonction avec la valeur -1.

Conseil : parler fort près du micro !

Lecture (fonction)

- **nf_sound_replay(n)** : lis le son nommé *Rn.wav* présent sur la carte SD.

Pour arrêter la lecture, utiliser la fonction avec la valeur -1.

Sons externes (fonction)

Il est possible de jouer des sons d'origine quelconque du moment qu'il respecte le format suivant :

- format wav non compressé
- fréquence d'échantillonnage : 8 kHz
- résolution : 8 bits non signés
- mono
- pas de metadonnées

Le logiciel Audacity permet de produire un tel fichier.

Ces fichiers doivent se nommer *Pn.wav* où n est un entier compris entre 0 et 32737.

- **nf_sound_play(n)** : lis le son nommé *Pn.wav* présent sur la carte SD.

Données sur carte SD

Il est possible d'écrire ou de lire des données sur une carte SD. Les données sont des entiers 16 bits (seul format autorisé par python pour Thymio) rangés dans une liste.

Fonctions

- **nf_sd_open(n)** : ouvre un fichier appelé *Un.DAT* en mode binaire lecture/écriture. Si ce fichier n'existe pas, il est créé. La valeur n doit être comprise entre [0:32767], en utilisant -1, on ferme le fichier courant ouvert.
- **nf_sd_write(data)** : écrit la liste de données *data* dans le fichier courant ouvert.
- **nf_sd_read(data)** : lit des données dans le fichier ouvert courant et remplit le tableau *data* avec.
- **nf_sd_seek(n)** : place la position de lecture à la position n dans le fichier. Attention, il n'y a pas de message d'erreur si vous dépasser la taille du fichier.

Il est possible de stocker un programme sur une carte SD. Voir la procédure en aseba dans la documentation en ligne. La transposition en python est assez aisée.

<https://aseba.wikidot.com>, menu documentation.



Auteur: JOËL RIVET

Session 1 - Annexe - API Thymio Python