Cours de Sciences de l'Ingénieur PeiP1

POL-E
le robot de Polytech Tours

Atelier de robotique

Guillaume Altmeyer [guillaume.altmeyer@univ-tours.fr]
Sébastien Bissey [sebastien.bissey@univ-tours.fr]
Remi Busseuil [remi.busseuil@univ-tours.fr]
Florent Chalon [florent.chalon@univ-tours.fr]
Jean-Paul Chemla [jean-paul.chemla@univ-tours.fr]
Pierre Gaucher [pierre.gaucher@univ-tours.fr]
Mathilde Gralepois [mathilde.gralepois@univ-tours.fr]
Sébastien Larribe [sebastien.larribe@univ-tours.fr]
Guénhaël Le Quilliec [guenhael.lequilliec@univ-tours.fr]
Matthieu Lescieux [matthieu.lescieux@univ-tours.fr]
Pascal Makris [pascal.makris@univ-tours.fr]
Nicolas Monmarché [nicolas.monmarche@univ-tours.fr]

12 février 2020

Table des matières

1 Présentation du document **6**

2 Présentation du robot **7**

I Conception mécanique **9**

3 Introduction **10**

4 Conception **11**
4.1 Matériaux 11
4.1.1 Classification 11
4.1.2 Grandeurs mécaniques carcatéristiques 12
4.1.3 Loi de comportement 12
4.2 Moyens de fabrication 14
4.3 DAO : Dessin Assisté par Ordinateur 15
4.4 Réalisation d'opérations élémentaires en DAO 15

5 Réalisation des pièces **16**
5.1 Impression 3D 16
5.1.1 Principe des imprimantes 3D FDM 16
5.1.2 Matériaux plastiques pour impression 3D FDM 19
5.1.3 Echange de données techniques 19
5.1.4 Logiciel de tranchage ou "slicer" 20
5.2 Découpage et gravure laser 21
5.2.1 Principe de la découpe et de la gravure laser 21
5.2.2 Principe de fonctionnement d'une machine de découpe laser 22
5.2.3 Matériaux pour la découpe et la gravure laser 22
5.2.4 Echange de données techniques 23
5.3 Références 23

II Programmation **24**

6 Introduction **25**

7 Mise en place **26**
7.1 Description de la carte WiPy 3.0 . . . 26
7.2 Carte d'extension pour WiPy 3.0 . . . 28
7.3 Ressources de la carte WiPy3 .0 – Identification . . . 29
7.4 Mise en service . . . 31
7.4.1 Mise en place de la carte WiPy3.0 sur la carte d'extension . . . 31
7.4.2 Connexion USB . . . 31
7.4.3 Mise à jour du *firmware* . . . 32
7.4.4 En cas de problème... . . . 32
7.5 Première connexion au Wipy . . . 33
7.5.1 Approche Windows . . . 33
7.5.2 Approche MacOs/linux . . . 35
7.6 Deuxième connexion au Wipy . . . 37

8 Quelques bases en python (micro-python) **40**
8.1 Introduction . . . 40
8.1.1 Les données . . . 40
8.2 Structures répétitives : les boucles . . . 41
8.3 Les fonctions . . . 42
8.4 Les classes . . . 42

9 Premiers tests **44**
9.1 Télécharger un programme . . . 44
9.2 Test de la LED . . . 45
9.2.1 Étapes . . . 45
9.2.2 Principales instructions . . . 45
9.2.3 Conclusion . . . 47
9.3 Plus de précisions sur le temps . . . 48

10 Comment capter la température... **49**
10.1 Le bus I2C . . . 49
10.2 Le capteur BME280 . . . 50
10.2.1 Présentation générale . . . 50
10.2.2 câblage . . . 51
10.3 Mise en œuvre . . . 51
10.3.1 Installation de la bibliothèque BME280 . . . 51
10.3.2 Mise en place de la connexion I2C . . . 56
10.3.3 Mise en place de la lecture du capteur BME280 . . . 56
10.4 Enregistrer les données sur une carte SD . . . 58
10.5 Conclusion . . . 60

11 Comment faire bouger son robot **61**
11.1 Introduction . . . 61
11.2 Principe de fonctionnement . . . 62
11.3 Mise en œuvre de connexions . . . 63
11.4 Test d'un moteur . . . 63

11.5 Mouvements du robot . . . 66
11.5.1 Améliorer les déplacements du robot . . . 66
11.6 Estimer la position et l'orientation du robot . . . 69
11.6.1 Principe de l'odométrie . . . 69
11.6.2 Mise en œuvre . . . 70

12 Comment détecter les obstacles 73
12.1 Introduction . . . 73
12.2 Mise en œuvre d'un capteur VL6180X . . . 73
12.2.1 Câblage . . . 75
12.2.2 Premier test du capteur . . . 75
12.3 Mise en œuvre des 3 capteurs VL6180X . . . 76

13 Intégration 79
13.1 Introduction . . . 79

III Électronique 80

14 Introduction 81

15 Fonctionnement d'une résistance et d'une DEL 82
15.1 La résistance . . . 82
15.1.1 Valeurs normalisées . . . 82
15.1.2 Code couleur . . . 83
15.2 La diode électroluminescente (DEL) . . . 83

16 Étude de la batterie 85
16.0.1 Tension de batterie . . . 85
16.0.2 Capacité de la batterie . . . 86

17 Etude de la propulsion 88

18 Caractérisation des capteurs 89

19 Principe de la soudure en électronique 90
19.1 Matériel nécessaire . . . 90
19.1.1 Le fer à souder . . . 91
19.1.2 La brasure ou fil à souder . . . 91
19.1.3 Nettoyeur de panne . . . 92
19.1.4 Éléments pour dessouder . . . 92
19.2 Sécurité . . . 93
19.3 Effectuer une soudure . . . 95
19.3.1 Mauvaises soudures . . . 96
19.4 Dessouder . . . 96
19.4.1 Avec une pompe à dessouder . . . 96
19.4.2 Avec une tresse à dessouder . . . 97

19.5 bibliographie 98

IV Aménagement-Environnement **99**

20 Introduction **100**

21 Importation des données collectées par le robot dans un SIG **102**

V Annexes **103**

A Liste du matériel **104**

B Mise à jour du *firmware* pour le Wipy **105**

C Ressources bibliographiques **107**

Introduction

Chapitre 1

Présentation du document

L'objectif de cet enseignement est de sensibiliser les élèves ingénieurs aux sciences de l'ingénieur au travers d'un projet multidisciplinaire mêlant l'informatique, l'électronique, la mécanique et l'environnement.

L'objectif de l'enseignement sera de concevoir un robot mobile, capable de se déplacer et d'effectuer des acquisitions de données. Le post-traitement des données recueillies sera ensuite effectué.

Les compétences acquises à travers cet enseignement seront : l'initiation à la gestion de projet, la gestion d'un projet pluridisciplinaire, des notions dans les domaines disciplinaires informatique, électronique, mécanique, et environnement.

Ce document est divisé en 4 grande parties. Chaque partie est indépendante et les parties peuvent être étudiées dans n'importe quel ordre.

Chapitre 2

Présentation du robot

Le robot POL-E est un robot mobile programmable en python qui est capable de percevoir des données environnementales et de les enregistrer pour un traitement ultérieur.

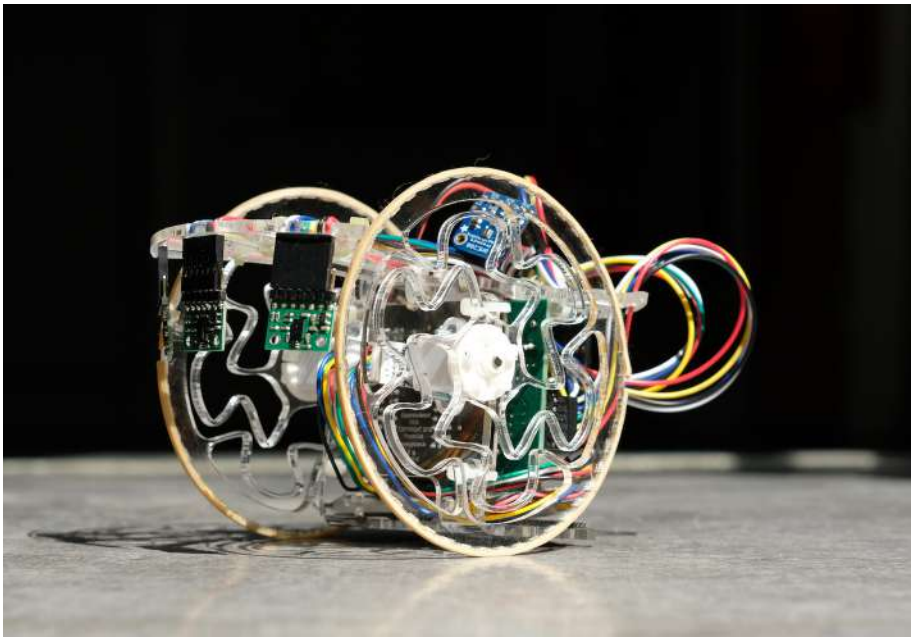


FIGURE 2.1 – Robot POL-E (version 2018).

La figure 2.2 montre le type d'environnement dans lequel les robots évolueront : des obstacles (représentés en rouge) doivent être évités. Les limites du terrain doivent également être prises en compte par les robots. Ces derniers se déplacent en évitant également les autres robots. Chaque robot effectue des mesures (ici, nous représentons une mesure de température).

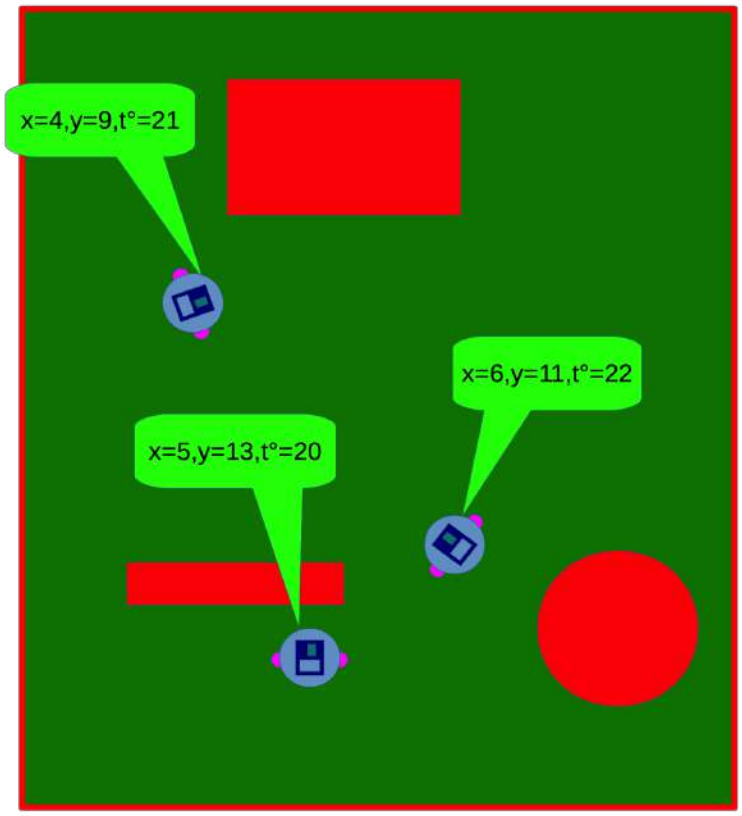


FIGURE 2.2 – Schéma général de fonctionnement des robots.

Première partie

Conception mécanique

Chapitre 3

Introduction

La partie mécanique de ce projet a pour objectif de concevoir et assembler les parties structurantes du robot. Pour ce faire, il faut mener une reflexion sur les matériaux à utiliser et le dimensionnement des pièces afin que le robot puisse recevoir tous les éléments nécessaires (moteurs, capteurs...) à son bon fonctionnement.

Chapitre 4

Conception

Pour réaliser un assemblage mécanique tel qu'un robot, il est nécessaire de procéder à une reflexion en plusieurs étapes. La prmeière de celles-ci est de prendre connaissance du cahier des charges imposé. Il faudra ensuite reflechir au dimensionement du système qui conduira ensuite au nombre de pièces à réaliser et au dimensionement de chacune des pièces qui composent le robot. En parallèle, il est nécessaire de s'interroger sur les matériaux dédiés aux diverses pièces en fonction de leurs roles dans le système mécanique. Enfin, la fabrication des pièces et l'assemblage sont réalisés et les tests préliminaires sont effectués.

4.1 Matériaux

Bref rappel sur les matériaux et leurs comportements

L'ensemble des matériaux utilisés dans la fabrication d'objets ou de systèmes et trés vaste et diversifié. On peut être amené à utiliser des matériaux très mous, tels que les caoutchoucs et les élastomères, des matériaux plus durs mais peu résistants aux fortes sollicitation mécaniques tels que les plastiques ou encore les métaux.

Nous allons donner ici une définition plus scientifiques des matériaux en citant leur classe, leur caractéristique et leur comportement mécanique.

4.1.1 Classification

Les matériaux peuvent être de diverses origines : minérale, organique, issue du pétrole... On peut classe ces matériaux en grandes 4 grandes catégories.

- Métaux et alliages : Fer, Aciers, Aluminium, Cuivre, Nickel, Titane... et les alliages. Les alliages de métaux sont des métaux dans lesquelles on incorpore des éléments (métalliques ou non) afin d'améliorer certaines caractéristiques.
- Céramiques et verres : Carbure de silicium (SiC), Nitrure de silicium (Si3N4), Alumine (Al2O3), Silice (SiO2), Silicates...
- Polymères : Polychlorure de vinyle (PVC), Polyéthylène (PE), Polyméthacrylaye de méthyle (PMMA), Nylon, Polystyrène (PS), Polyuréthane (PUR), Caoutchoucs...
- Composites : Fibre de verre, Fibre de carbones, Polymères chargés, Biocomposites, Bétons armés...

Ces matériaux possèdent des caractéristiques très différentes et sont utilisés en fonction de leur propriétés mécaniques les plus pertinentes en rapport avec l'utilisation visée. D'autres

caractéristiques sont également utilisées en dehors de la mécanique : isolation éléctrique et/ou thermique, corrosion...

Dans la section suivantes nous allons aborder les grandeurs caractéristiques des matériaux du point de vue mécanique.

4.1.2 Grandeurs mécaniques carcatéristiques

Nous allons ici définir les grandeurs les plus répandues pour caractériser les matériaux. Certaines de ces grandeurs sont également illustrées dans la section suivante.

- Masse volumique : quantité de matière par unité de volume (kg/m3)
- Dureté : Résistance aux marquages (rayures, empreintes...)
- Rigidité ou élasticité : Capacité d'un matériau à subir des déformations réversibles et proportionnelles aux contraintes
- Limite d'élasticité élasticité : valeur à partir de laquelle le matériau subit des déformations irréversibles
- Résistance au fluage : Capaité d'un matériau à se déformer peu lorsqu'il est soumis à une contrainte constante (fluage)
- Résistance à la fatigue mécanique :capacité d'un matériau à voir ses propriétés mécaniques peu modifiées suite à une sollicitation mécanique répétée
- Plasticité : déformation irréversible
- Thermo-plasticité : Déformation à la chaleur
- Formabilité : Aptitude d'un matériau à subir des déformations (pressage, emboutissage, laminage...)

4.1.3 Loi de comportement

En mécanique, la réponse mécanique d'un matériau face à des sollicitations mécaniques peut se caractériser par sa loi de comportement. Pour déterminer le comportement d'un matériau on a recours à des essais. Le plus répandu et le plus simple est l'essai de traction. Cela consiste à appliquer une force de traction à chacune des extrémités de l'éprouvette et de mesurer l'allongement en fonction de cette traction. On obtient ainsi une courbe de la force en fonction de l'allongement. La plus simple de celle-ci est la loi de Hooke ou loi d'élasticité (cf. figure 4.1).

Sur figure la 4.1 la courbe présente la force en fonction de l'allongement. En mécanique les grandeurs mécaniques s'exprime avec des variables différentes à savoir la contrainte, σ et la déformation, ε. La contrainte s'obtient, dans le cas d'un essai de traction qui subite de petites perturbations, en divisant la force par la section de l'éprouvette. L'unité de la contrainte est donc le Pascal. La déformation quant à elle s'obtient en divisant l'allongement de l'éprouvette par sa longueur initiale. La déformation est donc sans unité. La courbe obtenue est donc équivalente à celle de la figure 4.1. Le rapport entre la contrainte et la déformation se caractérise par un coefficient appelé le module d'élasticité ou le module d'Young E. La relation suivante appelé la loi de Hooke traduit la linéarité ou l'élasticité du matériau :

$$\sigma = E\varepsilon$$

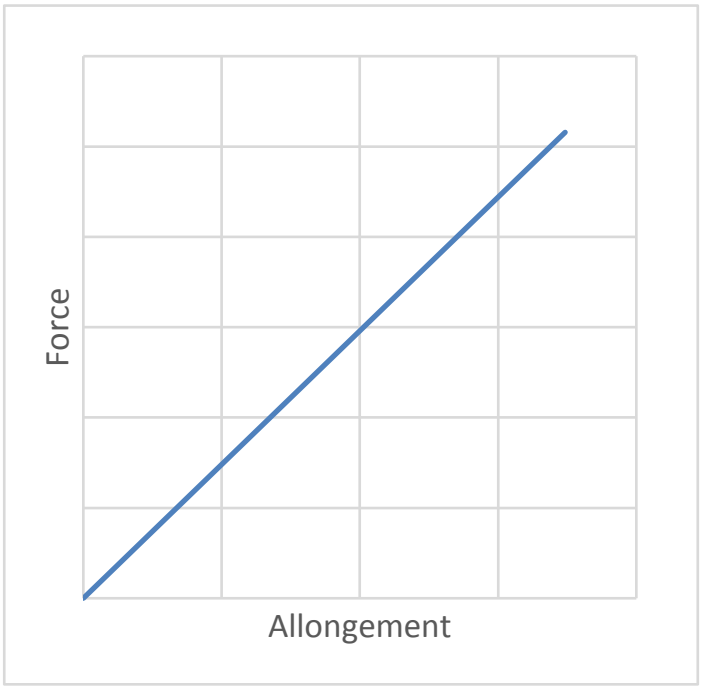


FIGURE 4.1 – Comportement linéaire

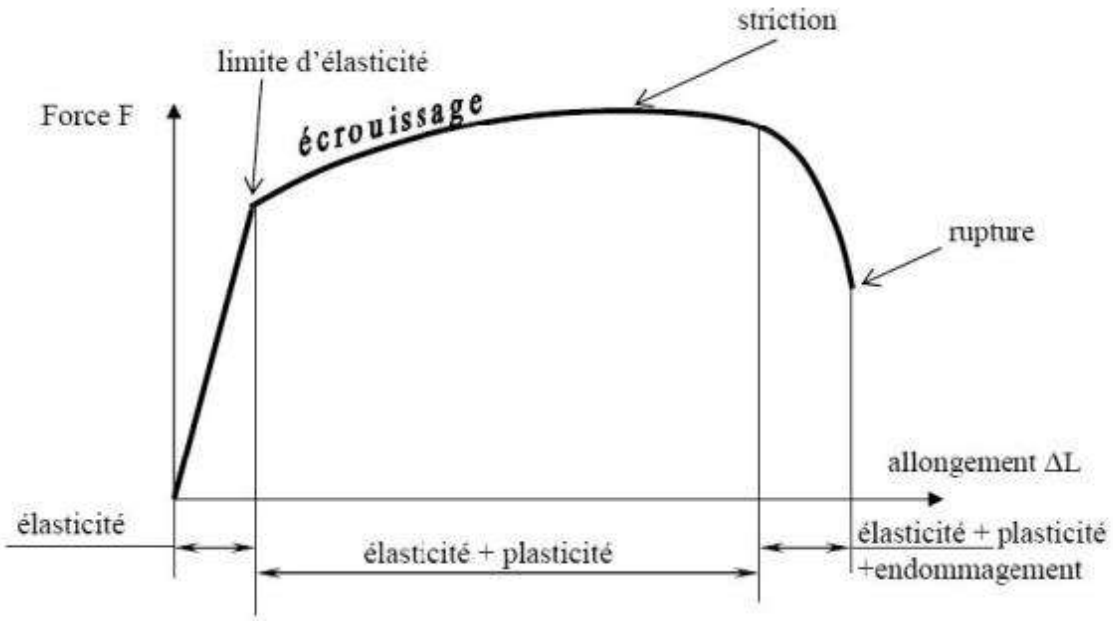


FIGURE 4.2 – Comportement élasto-plastique

L'unité du module d'Young est le Pascal.

Pour caractériser un matériau dans sa globalité, il est nécessaire de le solliciter au delà de sa partie linéaire. L'essai pratiquer est toujours un essai de traction mais on impose une force jusqu'à rupture du matériau. On obtient alors (dans le cas des métaux) la courbe sur la figure 4.2. Sur cette courbe on observe la partie élastique puis, à partie de la limite d'élasticité, la partie plastique du comportement du matériau. La partie advenant après la partie linéaire correspond à l'écrouissage du matériau ie sa capacité à se déformer de façon irréversible. Cette phase d'écrouissage peut être plus ou moins importante. Lorsque la déformation dans le domaine plastique est importante on parle alors de matériau *ductile*. Das le cas contraire on parle de matériau *fragile*.

4.2 Moyens de fabrication

Pour réaliser un système mécanique, il est nécessaire de procéder à la fabrication de chacune des pièces qui le compose. En ingénierie, les moyens de fabrication sont diverses et variés. Le moulage, l'usinage, le frittage (compaction de poudre par chauffage) et beaucoup d'autres procédés existent. Dans le cadre de ce cours, nous allons porter notre attention sur les moyens de fabrication disponibles dans l'école et que vous serez amenés à utiliser lors de ce projet ou de vos futurs projets : il s'agit ici de l'impression 3D et du découpage laser.

L'école est dotée d'un laboratoire de fabrication (FabLab) denommé Plug and Fab (PaF). Au sein de cet espace sont à dispositions des étudiants tout le matériel standard tel que l'outillage classique (tournevis, pinces, visseuse...) ainsi que des imprimantes 3D et une découpeuse

laser. Les imprimantes 3D permettent de fabriquer des objets par ajout de matière. Ce procédé est également nommé fabrication additive. La découpe laser est un procédé de découpe à l'aide d'un faisceau laser. Le fonctionnement et l'utilisation de ces deux appareils vous seront expliquer dans le chapitre 5. Dans le cadre de notre projet l'impression 3D et la découpe laser seront mise en oeuvre avec des matériaux plastiques (exemple : plexiglass) et polymères (exemple : Acrylonitrile Butadiène Styrène, ABS).

4.3 DAO : Dessin Assisté par Ordinateur

Une fois que la réflexion a été menée sur le dimensionnement des pièces, il faut alors les dessiner. Pour ce faire on a recours à des logiciels de Conception Assistée par Ordinateur (CAO). Ces logiciels permettent de réaliser toutes les étapes nécessaires avant de lancer la fabrication d'un produit. Dans le cadre de notre cours nous utiliserons en premier lieu le module de Dessin Assistés par Ordinateur (DAO) .Ensuite nous pourrons utiliser le module d'assemblage afin de positionner les pièces les unes par rapport aux autres pour réaliser un système mécanique tel que celui visé à savoir un robot.
Parmi les logiciels de CAO les plus répandu on trouve : Catia, SolidWorks, TopSolid, Autocad... Pour notre étude, Catia sera utilisé. La philosophie de fonctionnement des logiciels de DAO est la suivante. Lorsque l'on souhaite dessiner une pièce, la première étape est d'effectuer ce que l'on appelle une *esquisse*. Il s'agit de dessiner dans un environnement en 2 dimensions ie un plan choisi, la base d'une pièce. Par exemple, pour effectuer un cylindre, il est nécessaire de dessiner un cercle. On se place donc dans le plan désiré et on dessine ce cercle, ici l'esquisse, du cylindre du diamètre désiré (cf. figure ? ?). On quitte ensuite l'atelier de l'esquisse pour travailler dans un environnement en trois dimensions afin de pouvoir renseigner la hauteur du cylindre. Cette opération correspond à une *extrusion* (cf. figure ? ?). Pour les opérations les plus élémentaires, il conviendra donc de dessiner des esquisses sur des surfaces puis de s'appuyer sur celles-ci pour dessiner la pièce dans sa globalité. Ces opérations vous seront présentées lors des séances de cours.
Lorsque l'on travaille avec Catia et qu'on l'on souhaite sauvegarder un dessin, l'extension de du fichier associé est **.CATPart*. Cette extension est propre au logiciel Catia et peut être lu par certains logiciel. Afin de pouvoir utiliser les fichiers créés avec la découpeuse laser et l'imprimante 3D (cf.chapitre 5), il sera nécessaire de sauvegarder les dessins dans un autre format. Ceci se fait aisément avec Catia où il suffit de choisir l'extension de fichier appropriée.

4.4 Réalisation d'opérations élémentaires en DAO

Cette partie vous sera présentée lors de la première séance de mécanique.

Chapitre 5

Réalisation des pièces

Parler ici des moyens de fabrication présenter rapidement dans la section Moyens de fab. Ie fonctionnement de l'impression 3D : pré-traitement du fichier issu de la DAO, pilotage de l'imprimante, fonctionnement... Idem avec la découpe laser

5.1 Impression 3D

L'impression 3D recouvre plusieurs méthodes toutes basées sur la notion d'impression par couches successives d'un objet en 3 dimensions. Nous ne présenterons ici que l'impression par dépôt de matière ou FDM (pour Fused Deposition Modeling). Pour avoir un aperçu des autres types d'imprimantes 3D, vous pouvez consulter cette page : `http://www.primante3d.com/principe/`. La plupart des imprimantes FDM fonctionnent avec différents types de filaments en plastiques. Dans cette partie, nous présentons le principe général des imprimantes 3D, les matériaux plastiques les plus courants, le format de fichier STL qui comprend le modèle 3D à imprimer, le rôle d'un logiciel trancheur et les réglages de ce logiciel et le format de fichier gcode qui sera exécuté par l'imprimante.

5.1.1 Principe des imprimantes 3D FDM

Ces imprimantes 3D vont déposer de la matière (plastique fondu), couche par couche, de la couche la plus basse jusqu'à la couche la plus haute. Le principe de fonctionnement est illustré sur le schéma de la figure ci-dessous. Voici une liste non exhaustive des éléments qui composent une imprimante 3D :

- un plateau sur lequel l'impression va se faire. Ce plateau est souvent chauffant.
- 3 axes pour déplacer la tête d'impression dans les 3 dimensions au dessus du plateau
- un moteur qui commande l'extrusion du plastique (l'avancée du fil plastique)
- un élément chauffant qui permet de faire fondre le plastique avant son dépôt.
- une buse avec un trou assez petit (souvent 0.4mm) pour faire sortir le plastique fondu et le déposer.
- souvent un ventilateur pour refroidir rapidement le plastique qui vient d'être déposé.

Vu le principe de fonctionnement de l'imprimante FDM (dépôt par couches successives), il y a des points clés :

Accroche du plateau : la première couche imprimée a une grande importance : son rôle est

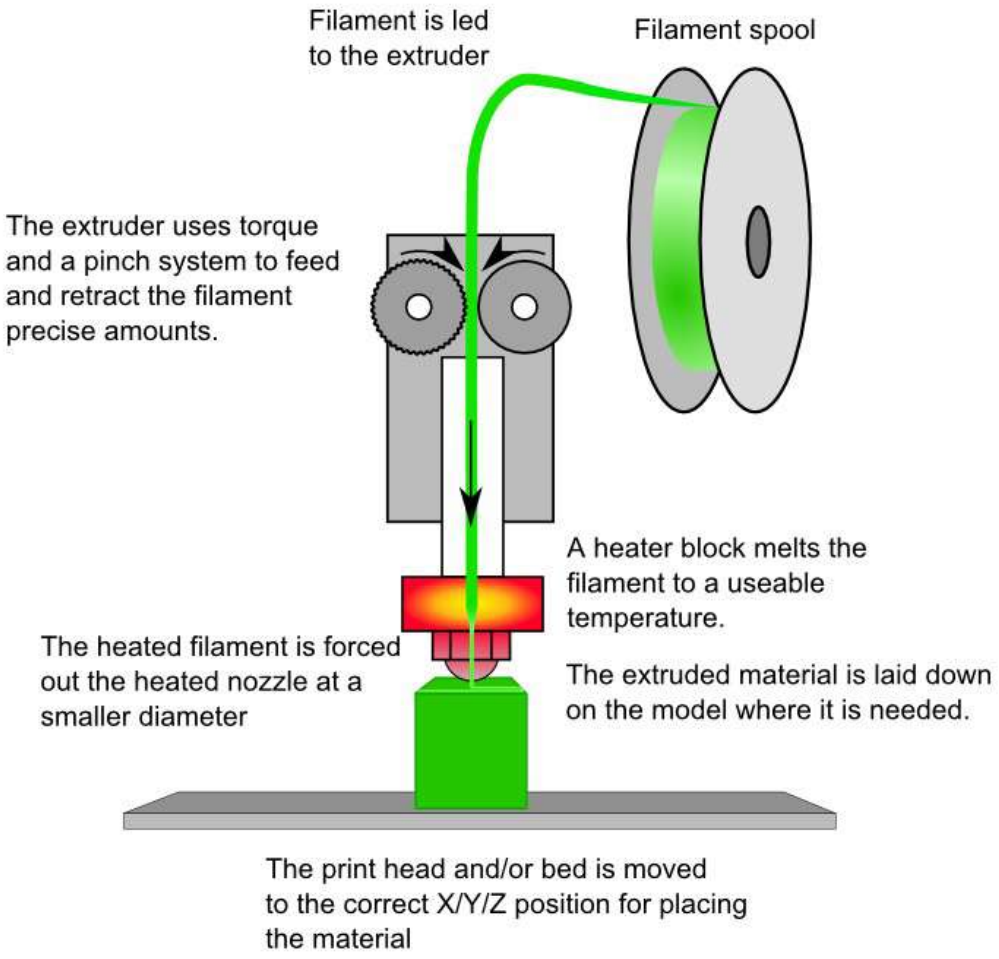


FIGURE 5.1 – Principe des imprimantes FDM (source : https ://www.thingiverse.com/thing :29432)

FIGURE 5.2 – ce petit bateau illustre la possibilité d'impression en léger surplomb et les ponts

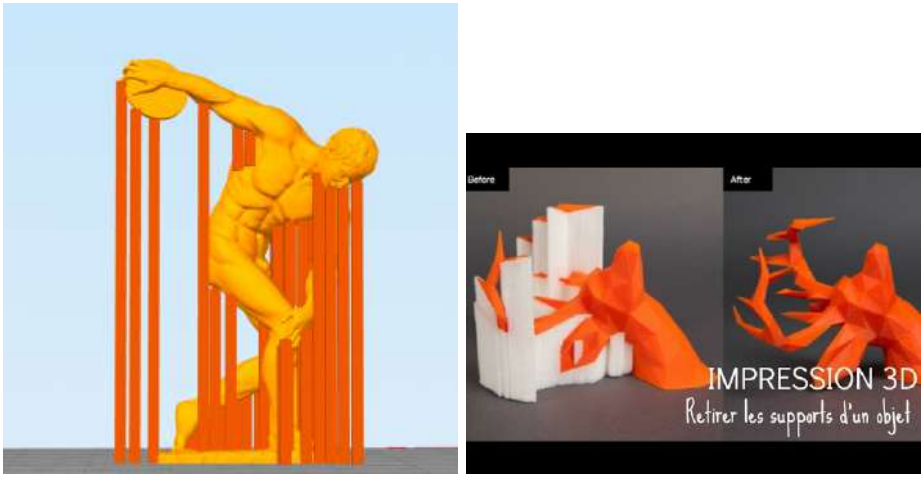


FIGURE 5.3 – Exemples d'ajout de support pour imprimer

d'assurer la cohésion de l'objet qui sera imprimé au plateau.Si l'objet se décolle, l'impression des couches supérieures ne seront plus positionnées par rapport au bas de l'objet (voir `http://giant.gfycat.com/ShadowyFamousGraysquirrel.gif`). Il faut aussi que cette couche ne soit pas trop petite par rapport à l'objet imprimé. Cela doit guider la façon dont on positionnera l'objet à imprimer sur le plateau.

Support : Pour imprimer les couches supérieures à la première couche,le plastique fondu doit être déposé sur les couches du dessous. un petit décalage en surplomb est possible. De même, faire un pont entre deux piliers est possible également, si le pont n'est pas trop grand. Sur l'image du petit bateau imprimé ci-dessous, on voit l'illustration de ces deux possibilités (le pont est par exemple le dessus de la fenêtre carrée). Par contre, commencer une partie d'un objet qui ne repose sur rien est impossible sans support. Le support est une partie imprimée qui doit être retirée. Certaines imprimantes permettent d'imprimer 2 matériaux distincts. Dans ce cas, un des matériaux peut être un plastique soluble utilisable pour créer le support. Sinon, le support doit être retiré mécaniquement. La figure ci-dessous montre comment sont ajoutés les supports pour rendre l'impression possible.

Non homogène La cohésion des couches vient du fait qu'en déposant le plastique fondu sur la couche précédente, il y aura une continuité de matière. Mais cette cohésion

entre couche n'a pas la même tenue mécanique que chaque couche. Ainsi il vaut mieux imprimer un anneau à l'horizontal (à plat) qu'à la verticale.

5.1.2 Matériaux plastiques pour impression 3D FDM

Au PAF de l'école nous trouverons 3 plastiques différents :

PLA pour Polylactic acid. C'est un bioplastique facile à imprimer et est un des matériaux les plus courants pour l'impression 3D. Le plateau chauffant n'est pas nécessaire et il ne dégage pas d'odeur désagréable. Son inconvénient principal est de ne pas tenir les températures un peu élevées ($50^o, 60^o$ maximum).

ABS ou acrylonitrile butadiène styrène. C'est un plastique courant obtenu à partir du pétrole. Il fond à plus haute température et est plus délicat à imprimer : un plateau chauffant et une enceinte fermée sont nécessaires sinon la pièce peut se soulever sur les bords (wrapping) pour finir par se décoller complètement du plateau. L'impression d'ABS dégage une odeur désagréable.

PVA ou polyvinyl alcohol. C'est un plastique soluble à l'eau chaude. Son utilisation se limite à la réalisation de support. Toutes les bobines de fil sont sensibles à l'humidité mais celles-ci bien plus que les autres.

Les matériaux sont vendus en bobines (sous vides avec des sachets de dessicants) d'1kg environ. Il existe du fil de 1,75mm de diamètres (pour les petites imprimantes du PAF) et du 2,85mm pour la 3NTR, la HP et l'Ultimaker 3.

Deux sites avec l'ensemble des plastiques utilisables dans les imprimantes 3D :

`https://www.filimprimante3d.fr/documents/memento-materiau-impression-3d.pdf`

`https://www.simplify3d.com/support/materials-guide/properties-table/`

Sur ce dernier site, vous trouverez les propriétés mécaniques (évoqués au chapitre précédents) des matériaux pour l'impression 3D.

5.1.3 Echange de données techniques

Différentes étapes sont nécessaires pour transférer les données techniques des phases de de conception jusqu'au'à la construction de la pièce finale.

La conception assistée par ordinateur permet au concepteur de traduire son idée vers un modèle numérique basé sur les informations géométriques du produit. Ces informations géométriques regroupent des informations telles que les coordonnées des points, les points par lesquels passent les courbes, les ensemble de courbes définissant les surfaces des pièces et enfin les ensemble de surfaces limitant le volume de la pièce. Différentes syntaxes peuvent être utilisés pour générer les fichiers de pièces et assemblages, permettant ou non la lecture directe de ces informations dans un fichier texte. Certains formats fermés sont propriétaires et associés à un logiciel donné, par exemple des fichiers *.CATPart pour les fichiers générés sous Catia, ou au contraire sont plus ouverts pour favoriser l'échange de données géométriques entre les logiciels, comme les fichiers *.stp ou *.igs.

Une autre manière de transmettre les informations sur la géométrie de la pièce est s'utiliser un maillage de sa surface. La surface de la pièce est alors discrétisée à l'aide d'un

nombre fini de points. Ces points, ou nœuds, reliés entre eux forment des triangles qui s'approchent de la surface de la pièce initiale. La qualité de l'approximation dépend de la complexité géométrique de la pièce et du degré de finesse du maillage, donc du nombre de noeuds utilisés lors de la discrétisation. Les noeuds et leur connectivité peuvent ensuite être stockées dans des fichiers. Le format de stéréolithographie ou *.stl, développé initialement par la société 3D System, s'est imposé comme format de référence. La plupart des logiciels de CAO sont actuellement dotés de fonctions d'export vers le format *.stl et permettent de générer des fichiers de données destinés à la fabrication numérique. Il est à noter que les algorithmes de génération de ces maillages sont différents de ceux utilisés par les mailleurs destinés au calcul de structures, les objectifs et contraintes de ces deux dommaines n'étant pas les mêmes. Ce format permet l'échange de fichiers 3D (voir `http://www.thingiverse.com` mais avec peu de possibilité de modification de conception (on a le maillage sans description de la géométrie de la pièce).

La géométrie de la pièce, définie par son maillage au format *.stl, est ensuite convertie dans un autre format définissant le trajet de l'outil permettant de produire cette pièce. Pour une pièce fabriquée par impression 3D, cette opération consiste à découper la pièces en couches successives et à définir pour chaque couche le trajet que doit suivre la buse pour déposer la matière. Cette opération est effectuée par un logiciel appelé (en anglais) "slicer". Les caractéristiques de chaque machine doivent être prises en compte par le logiciel pour la définition du trajet de l'outil. Chaque machine est donc livrée avec un logiciel paramétré permettant de définir le trajet optimal de la buse. Des logiciels libres puissants et hautement paramétrables sont aussi disponibles.

La dernière étape consiste à traduire les trajectoires des outils en langage compréhensible par la machine à commande numérique. Un langage actuellement largement répandu est le G-code. Le G-code est composé d'opérations élémentaires permettant d'initialiser l'imprimante, de régler la position initiale de la buse ou des outils, de définir leur déplacement et la vitesse d'avance dans l'espace. La liste de ces commandes élémentaires permettant d'écrire un programme en G-code peut par exemple être consultée sur : `https://fr.wikipedia.org/wiki/Programmation_de_commande_numerique` . De nombreux logiciels de CFAO propriétaires ou libres sont disponibles pour générer le G-code.

L'envoi vers l'imprimante 3D du G-code produit à partir du modèle géométrique de la pièce permet de lancer la phase d'impression de la pièce.

5.1.4 Logiciel de tranchage ou "slicer"

Les logiciels permettant de passer d'un fichier *.stl à un fichier .gcode sont appelés logiciel de tranchage. Il en existe plusieurs gratuits dont Slic3r et Cura. Le fichier *.gcode décrit les trajectoires pour chaque couche de l'imprimante. Ce fichier contiendra la liste des déplacements, la commande de l'extrudeur et les réglages de température. Il va dépendre de l'imprimante 3D, du matériau utilisé et des réglages choisis dans le logiciel de tranchage. Voici les principaux réglages que l'on trouve dans ces logiciels.

Positionnement : on place l'objet décrit dans le fichier *.stl sur le plateau : orientation et éventuellement redimensionnement.

Choix dépendant du matériaux : température de l'extrudeur, température du plateau. Ces valeurs sont souvent indiquées sur l'emballage des bobines.

Qualité d'impression : La hauteur de la couche d'impression, en général de 0,1mm à 0,3mm. Une couche plus fine permet de gagner en qualité mais au détriment du temps d'impression. Les vitesses d'impression dépendent de la capacité de l'imprimante à accélérer la tête d'impression (une structure rigide permet une impression plus rapide).

Taux de remplissage : un faible remplissage permet une impression plus rapide et un gain en matière, au détriment de la solidité mécanique. On peut choisir également l'épaisseur des parois.

Adhérence du plateau : Si l'adhérence de l'objet sur le plateau pose problème, On peut demander une bordure (brim) de quelques mm ou un radeau (lit imprimé sous l'objet) pour augmenter cette adhérence.

5.2 Découpage et gravure laser

En complément des procédés additifs, permettant de créer des pièces par ajout de matière, de nombreux procédés de fabrication sont basés sur l'enlèvement de matière. Parmi ces procédés soustractifs, l'usinage permet d'obtenir des pièces finales après enlèvement de matière sur un brut. Les trous sont obtenus par perçage, les pièces de révolution par tournage, alors que des formes complexes peuvent être obtenues par fraisage. Les procédés de découpe appartiennent aussi à cette catégorie. Ils sont particulièrement adaptés à la création de pièces planes. La découpe peut être effectuée par cisaillement, par oxycoupage, par plasma, par jet d'eau ou encore par laser. Le choix d'un procédé dépend du matériau, de l'épaisseur de la tôle ou encore de la précision souhaitée. Les pièces finales aux formes 3D complexes peuvent ensuite être obtenues par pliage ou par emboutissage. Pour la création de robots dans le cadre de ce projet, l'utilisation de la découpe laser est particulièrement adaptée à la création de pièces de structures. Ce procédé sera donc détaillé dans cette section en se focalisant sur les moyens disponibles au PAF.

5.2.1 Principe de la découpe et de la gravure laser

Une source laser émet un rayonnement lumineux cohérent concentré en un point à l'aide d'une lentille. La concentration du flux permet en ce point précis une forte élévation de la température et un changement d'état de la matière. Suivant les matériaux utilisés, une vaporisation ou une combustion peuvent être observées. Il en résulte un enlèvement de matière très localisé. Le déplacement de la buse de focalisation est effectué dans un plan parallèle à la tôle. L'assistance par commande numérique permet de définir des trajets de coupe à partir de modèles CAO 2D et d'obtenir des pièces planes tout en respectant des côtes précises sur tout le contour de la pièce. La concentration de l'énergie permet de n'affecter thermiquement qu'une zone dont la largeur n'excède généralement pas quelques centaines de microns. Le faible volume de matière impactée et sa localisation près des bords de la pièce permettent d'éviter l'introduction de contraintes résiduelles dans le cœur de la pièce. La gravure laser est basée sur le même principe que la découpe et utilise le même type d'équipements. La principale différence vient de la profondeur de la zone à traiter et de la définition du trajet de la buse. Si certains équipements de faible puissance sont généralement dédiés à la gravure laser, la

diminution de puissance ou l'augmentation de la vitesse de la buse d'une machine de découpe laser permettent d'effectuer des opérations de gravure.

5.2.2 Principe de fonctionnement d'une machine de découpe laser

La machine de découpe laser se composent des éléments suivants :
- Carter pour protéger l'utilisateur du rayonnement et d'émissions nocives. Cet élément n'est pas systématiquement présent sur les machines de faible puissance.
- Cadre métallique pour supporter les tôles à découper.
- Source d'émission laser pulsé (YAG) ou continu (CO2) pour générer le flux lumineux nécessaire à la coupe.
- Système de miroirs pour transmettre le faisceau laser jusqu'à la tête laser.
- Tête laser équipée d'une lentille réglable pour focaliser la lumière précisément sur le point à découper.
- Moteurs pas à pas et glissières pour positionner la buse et la déplacer sur le long du trajet de coupe.
- Carte mère pour piloter la source d'émission laser, la buse et les moteurs.
- Boitier de puissance pour alimenter les différentes parties de la machine.
- Extracteur d'air et filtres, permettant de collecter et filtrer les émissions.
- PC pour générer le code à partie d'un fichier CAO et contrôler la machine.

5.2.3 Matériaux pour la découpe et la gravure laser

La découpe laser est largement utilisée pour la découpe de matériaux métalliques, notamment d'aciers ou d'alliages d'aluminiums, et de polymères. La puissance de la source laser de la machine de découpe du PAF étant de quelques dizaines de Watts, seules les tôles en polymères peuvent être traitées. Parmi les nombreux matériaux utilisés en découpe laser de polymères, il est possible de citer :

le PMMA, acrylique ou polyméthacrylate de méthyle (Plexiglas) : ce polymère thermoplastique est largement utilisé en découpe laser pour ses propriétés mécaniques, optiques, chimiques et esthétiques. D'un point de vue mécanique, il se révèle être léger, rigide, résilient, résistant aux UV et facilement usinable.

le PET ou polyéthylène téréphtalate.

le PA, polyamide ou nylon.

le PS ou polystyrène.

le PP ou polypropylène.

l'ABS ou acrylonitrile butadiène styrène.

le PUR ou polyuréthane : la découpe de plaques de mousses de polyuréthane permet de concevoir et fabriquer de manière très précise des pièces de protection légères. Il peut être utilisé pour le packaging, l'amortissement ou la protection contre les chocs.

le bois, le contreplaqué et le MDF : les plaques de bois de de ses dérivés sont des matériaux adaptés à la découpe laser. Le MDF est un matériau constitué à partir de fibres de bois réparties de manière aléatoires, prisé des artistes et modélistes. Lors du passage du faisceau laser sur la plaque, une combustion peut se produire, dégageant une odeur

de brûlé. Des traces brunes peuvent apparaitre dons la zone thermiquement affectée, créant des défauts esthétiques. Un réglage adéquat des paramètres de coupe ou une découpe multi-passe peuvent limiter le phénomène.

Il existe aussi des tôles bi-couches combinant un polymère et une fine feuille de métal, par exemple acrylique et aluminium, utilisées par les designers. Outre le matériau, l'épaisseur de la tôle est aussi un facteur déterminant lors du réglage des paramètres de la machine. Les tôles généralement utilisées au PAF ont des épaisseurs variant de 2mm à 8mm. Plus une tôle est épaisse pour un matériau donné, plus l'énergie nécessaire à la découpe est importante. Il est possible de déterminer celle-ci à partir de la puissance du laser choisie et du réglage de la vitesse d'avance de la tête de découpe.

En raison de propriétés optiques, d'émissions gazeuses nocives ou de risque d'inflammation, certains polymères ne peuvent au contraire pas être utilisés en découpe laser, comme par exemple :

les métaux polis, miroirs et polymères à surface réfléchissante : réflexion du faisceau laser

le PVC, poly chlorure de vinyle ou PVC : émission de gaz chloré

le PTFE, polytétrafluoroéthylène ou téflon : émission de gaz fluoré

5.2.4 Echange de données techniques

Tout comme pour les opérations d'impression 3D, l'utilisation de formats de fichiers informatiques adaptés est nécessaire pour commander la machine de découpe laser. La chaine numérique est toutefois rendue plus simple par la nature 2D de la géométrie à traiter.

La géométrie de la pièce est créée à partir d'un logiciel de dessin ou d'un modeleur CAO 2D ou 3D. Les informations nécessaires à la définition d'un trajet de coupe sont contenues dans la ou les esquisses 2D de la pièce. Un fichier, généralement au format .svg ou .dxf, créé à partir des esquisses permet donc d'exporter les informations géométriques.

Les esquisses générée par l'outil CAO sont exportée dans un format est ensuite traduit en trajet de coupe pour la machine. Cette opération de génération du Gcode peut être effectuée par le logiciel de la machine ou par des logiciels dédiées, propriétaires ou libres.

5.3 Références

Deuxième partie

Programmation

Chapitre 6

Introduction

Sous le terme « programmation » nous sous-entendons la partie commande du robot : il s'agit de programmer les actions du robot. Plus précisément, il s'agit d'envoyer des ordres aux moteurs pour que le robot se déplace, de lire les informations provenant des capteurs et d'en tirer des conséquences sur le déplacement du robot.

La partie du robot capable de remplir ces fonctions est le microcontrôleur. On peut assimiler ce composant à un cerveau dans le monde animal : il reçoit des informations et prend des décisions, donne des ordres de mouvement. Le microcontrôleur est similaire au microprocesseur d'un ordinateur, il est cependant plus simple et moins rapide ce qui a pour conséquence qu'il n'exécute qu'un seul programme à la fois.

Les microcontrôleurs sont le plus souvent programmés en langage C (ou C++). Nous avons cependant sélectionné un microcontrôleur programmable en Python car ce langage est souvent utilisé dans les lycées, de plus il est rapide à prendre en main par un débutant.

Les objectifs de cette partie sont :

— de configurer son environnement de travail pour pouvoir programmer le robot en Python ;
— de découvrir le fonctionnement d'un microcontrôleur à travers la carte Wipy3.0 ;
— de découvrir (ou de se perfectionner) le langage python ;
— d'écrire des programmes, de les tester, pour permettre au robot de se déplacer, d'utiliser ses capteurs.

La répartition (indicative) des 4 séances est la suivante :

1. mise en place et configuration des outils, test du premier programme permettant de contrôler les Leds de la carte. Prise en compte du capteur de température, stockage de ces informations avec les coordonnées du robot sur la carte SD ;
2. contrôle des moteurs et déplacement du robot. Écriture de fonctions facilitant l'enchaînement de déplacements du robot ;
3. prise en compte des capteurs de distance pour sécuriser les mouvements du robot ;
4. mise en place d'une stratégie de déplacement en fonction des informations des capteurs.

Chapitre 7

Mise en place

Nous utilisons une carte Wipy. Après une rapide présentation de cette carte et une liste de ces ressources (en particulier les possibilités de connexion avec des capteurs, moteurs...), nous présenterons la mise en place des outils logiciels permettant de programmer cette carte.

7.1 Description de la carte WiPy 3.0

Les principaux éléments de la carte microcontrôleur WiPy 3.0 sont décrits ci-dessous. Cette carte WiPy3.0 possède les caractéristiques suivantes :

Features

- *Powerful CPU, BLE and state of the art WiFi radio*
- *1KM WiFi range*
- *MicroPython enabled, the Linux of IoT for fast deployment*
- *Fits in a standard breadboard (with headers)*
- *Ultra-low power usage : a fraction compared to other connected micro controllers*

Processing

- *Espressif ESP32 chipset*
- *Dual processor WiFi radio system on chip*
- *Network processor handles the WiFi connectivity and the IPv6 stack*
- *Main processor is entirely free to run the user application*
- *An extra ULP-coprocessor that can monitor GPIOs, the ADC channels and control most of the internal peripherls during deep-sleep mode while only consuming 25uA*

Interfaces

- *2 x UART*[1]*, 2 x SPI*[2]*, I2C*[3]*, I2S, micro SD card*
- *Analog channels : 8_ 12 bit ADCs*
- *Timers : 4_ 16 bit with PWM*[4] *and input capture*

1. UART : *Universal Asynchronous Receiver Transmitter*
2. SPI : *Serial Peripheral Interface*
3. I2C : *Inter-Integrated Circuit*
4. PWM : *Pulse Width Modulation*

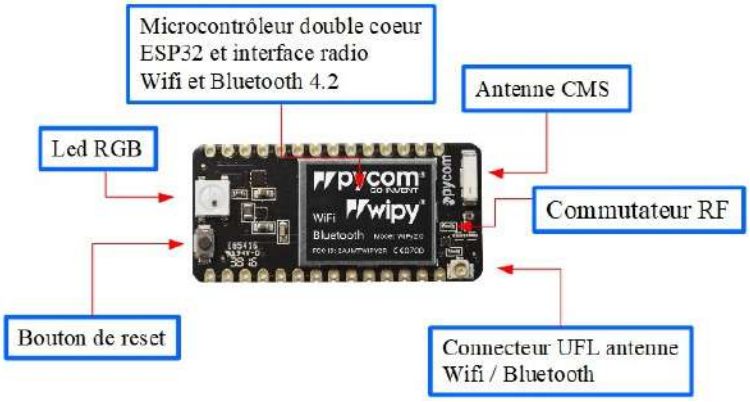


FIGURE 7.1 – Principaux constituants de la carte Wipy 3.0.

— *DMA on all peripherals*
— *GPIO : Up to 24*

Hash/Encryption
— *SHA, MD5, DES, AES*

WiFi
— *802.11b/g/n 16mbps*

Bluetooth
— *Low energy and classic*

RTC[5]
— *Running at 32KHz*

Power
— *3.3V to 5.5V*
— *3V3 output capable of sourcing up to 550mA*

Security & Certifications
— *SSL/TLS support*
— *WPA Enterprise security*
— *FCC 2AJMTWIPY3R*
— *CE 0700*

5. RTC : *Real Time Clock*

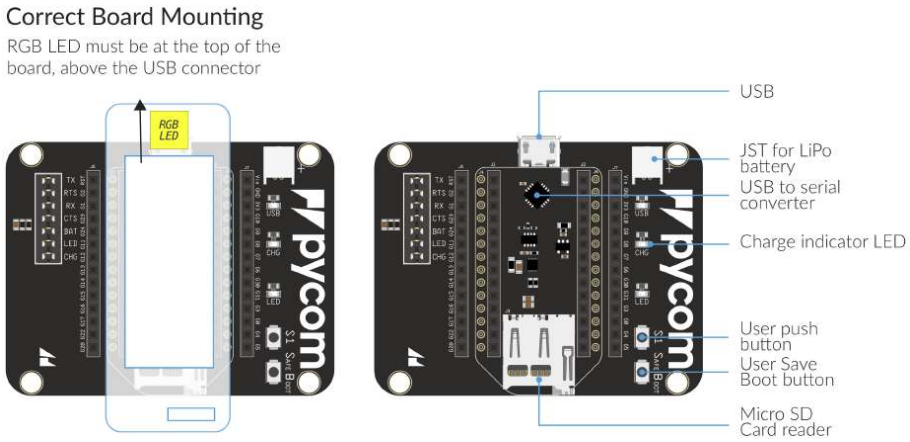


FIGURE 7.2 – Carte d'extension pour la carte WiPy3.0

Memory

- *RAM : 4MB*
- *External flash : 8MB*
- *Hardware floating point acceleration*
- *Python multi-threading*

Size

- *42mm x 20mm x 2.5mm (excluding headers)*

Weight

- *5g*
- *29g (Packaged)*

7.2 Carte d'extension pour WiPy 3.0

Cette carte d'extension (*Expansion Board 3.0*) permet de gérer les fonctionnalités décrites ci-après. Elle permet d'accueillir la carte WiPy 3.0.

Il existe d'autres cartes d'extension qui embarquent plus de capteurs : Pysense (*Ambient light sensor, Barometric pressure sensor, Humidity sensor, 3 axis 12-bit accelerometer, Temperature sensor*) et Pytrack (*Super accurate GNSS + Glonass GPS, 3 axis 12-bit accelerometer*).

Voici les principales caractéristiques de cette carte d'extension (qui s'ajoutent donc aux caractéristiques du Wipy 3.0) :

Features

- *USB and LiPo battery powered*
- *Custom PIC USB to serial converter with automatic bootloader mode*
- *LiPo battery charger (BQ24040), with options for two different charging currents (100mA and 450mA)*
- *TPS2115A with reverse voltage protection*

N° cavalier	Désignation broche	Fonction si cavalier en place	Fonction si cavalier retiré
1	Tx		
2	RTS		
3	Rx		
4	CTS		
5	BAT	Led témoin de charge active	Led témoin de charge désactivée
6	LED	Led active	Led inactive
7	CHG	Courant de charge : 450mA	Courant de charge : 100mA

TABLE 7.1 – Configuration des cavaliers de la carte d'extension *Expansion Board 3.0*.

— *MicroSD card slot*
— *JST style battery connector*
— *Power LED and charge status LED*
— *One user LED and one user switch*
— *Button to enter into "safe mode" easily*
— *Battery voltage monitoring via the WiPy ADC*
— *Lots of jumpers to enable/disable features*

Size & weight
— *Size : 65 x 50 x 8mm*
— *Weight : 18g, 43g (Packaged)*

La configuration des cavaliers sur la carte d'extension correspond aux fonctionnalités données par la table 7.1.

Remarque : attention, les cavaliers ont tendance à tomber, en cas d'anomalie il faut penser à vérifier qu'ils sont présents.

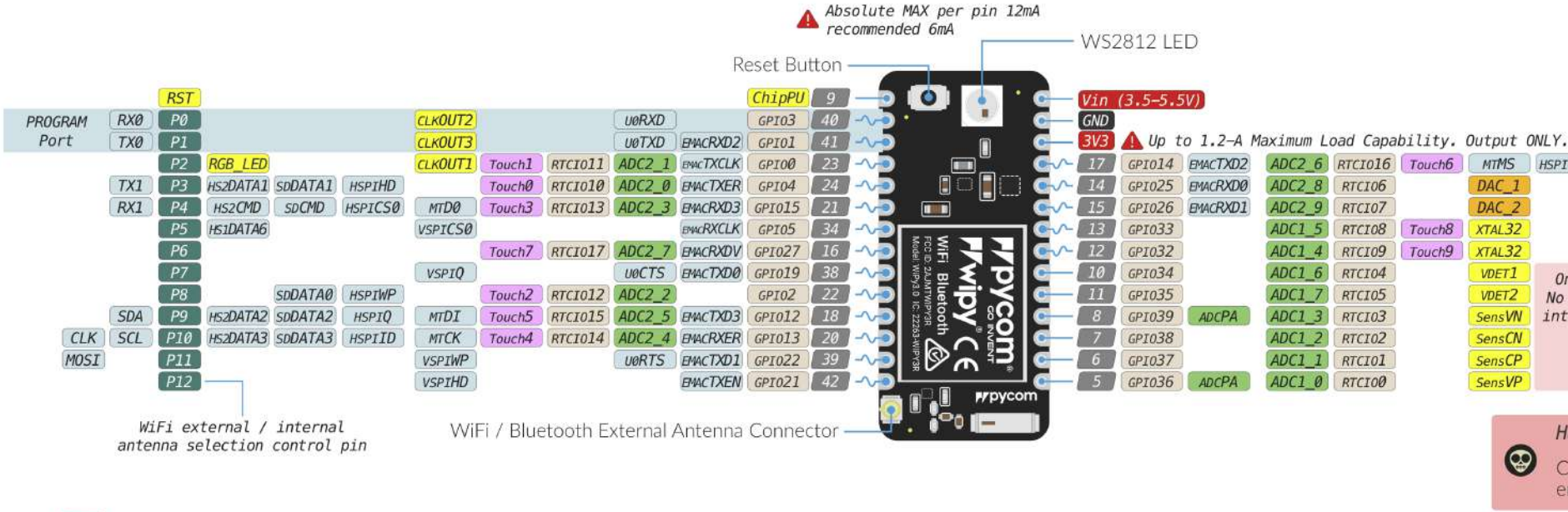
7.3 Ressources de la carte WiPy3 .0 – Identification

Les figures ci-dessous recensent les ressources de la cartes WiPy3.0 (cf. 7.3) ainsi que celles de la carte d'extension (cf. 7.4).

Le tableau **??** recense la correspondance entre la nomenclature des ports d'entrée-sortie (E/S) de la carte WiPy3.0, la désignation des GPIO [6] et la référence des broches (*pins*) de la carte d'extension.

6. GPIO : *General Purpose Input Output*, soit port général d'entrée-sortie

Model: WiPy3.0



- Power
- GND
- Serial Pin
- Analog Pin
- Control
- Physical Pin
- Port Pin
- Touch Pin
- DAC Pin
- PWM Pin

Low Level Bootloader

P2 + GND

Boot modes and safe boot

P12 + 3V3

1-3 sec *Safe boot, latest firmware is selected*

4-6 sec *Safe boot, previous user update selected*

7-9 sec *Safe boot, the factory firmware is selected*

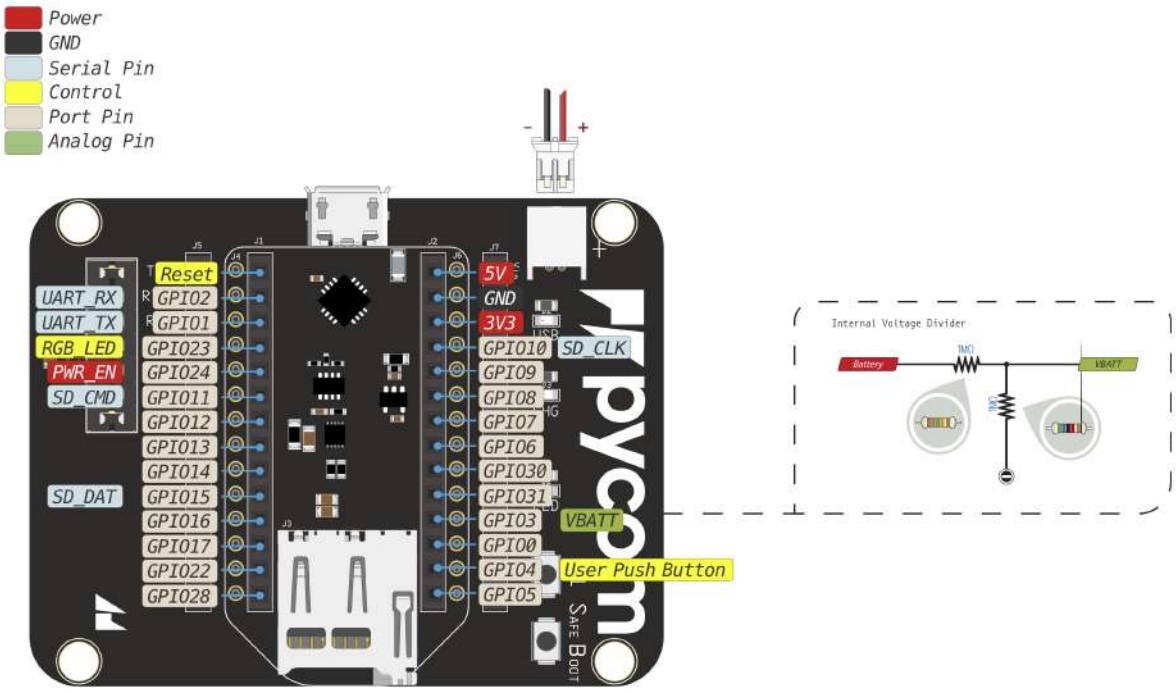


FIGURE 7.4 – Mappage des broches de la carte d'extension de la carte WiPy3 .0

7.4 Mise en service

7.4.1 Mise en place de la carte WiPy3.0 sur la carte d'extension

La carte WiPy3.0 doit être insérée sur la carte d'extension en respectant le sens défini sur la figure 7.5.

Attention, il faut bien aligner les broches et éviter de les tordre... Heureusement, une fois installée la carte Wipy n'a plus besoin d'être enlevée de sa carte d'extension. Elle resteront solidaires l'une de l'autre y compris sur le robot.

7.4.2 Connexion USB

Une fois la carte WiPy mise en place sur la carte d'extension, nous pouvons alors la connecter en utilisant le câble USB fourni. Après quelques secondes, la led RGB de la carte WiPy clignote (en bleu) toutes les 4 secondes, indiquant que tout fonctionne correctement. Par la suite, cela sera un bon moyen de vérifier que tout va bien du côté de la carte Wipy.

La connexion à l'ordinateur provoque des actions dépendantes du système d'exploitation utilisé. Sans rentrer dans les détails, l'ordinateur doit installer des pilotes de périphériques (*drivers*) permettant de faire communiquer l'ordinateur et la carte à travers le câble USB. Il est souvent nécessaire d'avoir des droits d'administration sur la machine pour que l'installation se passe bien.

Si tout va bien, la communication peut avoir lieu par intermédiaire d'un « port COM x » sous Windows (que l'on peut traduire par « canal de communication numéro x »). Sous

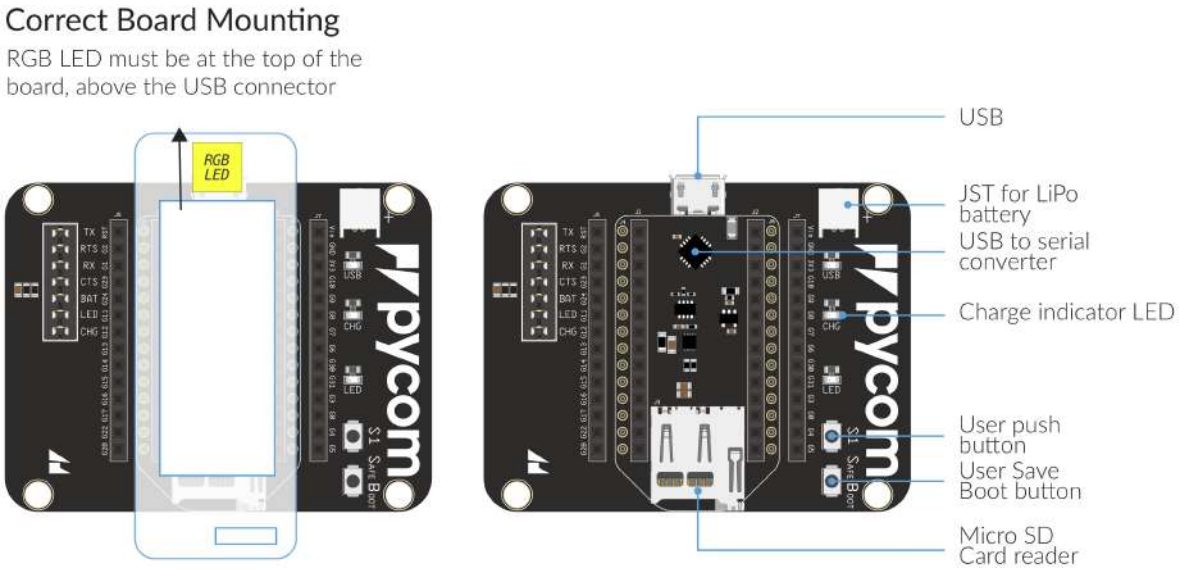


FIGURE 7.5 – Mise en place de la carte WiPy3.0 sur la carte d'extension

les systèmes MacOs, ce canal de communication se nomme : `/dev/tty.usbserial` (`dev` signifie *device* : matériel).

7.4.3 Mise à jour du *firmware*

Le *firmware* (que l'on pourrait traduire par « couche logicielle liée au fabriquant du matériel », c'est un peu long et on comprend que le terme firmware soit adopté par tout le monde), prend en charge le fonctionnement du Wipy : c'est le programme qui permet à l'ordinateur de rentrer en contact, de transmettre les programmes que vous allez écrire.... C'est en quelque sorte le système d'exploitation (miniature) qui permet au Wipy de fonctionner.

Il est possible que ce firmware necessite des mises-à-jour (pour corriger des problèmes ou bien ajouter des fonctionnalités). Le fabriquant encourage d'ailleurs cette mise à jour :

> *We strongly recommend you to upgrade your firmware to the latest version as we are constantly making improvements and adding new features to the devices.*

La procédure de mise à jour du *firmware* de la carte est décrite via le lien : `https://docs.pycom.io/chapter/gettingstarted/installation/firmwaretool.html`.

Le détail de cette opération peut-être trouvé en annexe de ce document.

7.4.4 En cas de problème...

Nous n'avons rien fait de concret mais les problèmes peuvent déjà être présents :

— l'installation des *drivers* sur l'ordinateur : problèmes de droits (pas le droit d'installer des nouveaux pilotes), pas le droit d'utiliser les pilotes (qui peuvent être correctement installés mais utilisables par un utilisateur plus chanceux), problème de version, de disponibilité des fichiers (qui ne sont pas toujours disponibles automatiquement),... ;

— problèmes matériels : le câble USB (le fil, les prises) peut-être défectueux (écrasement, coupure invisible...). La carte d'extension ou le Wipy peuvent avoir un défaut...

Pour les problèmes liés à l'ordinateur sur lequel vous allez programmer, la résolution des problèmes éventuels nécessite que vous soyez attentifs aux messages du système (qui ne sont pas toujours visibles...). L'enquête commence par ces indices. Souvent, le problème peut être résolu avec l'aide de quelqu'un ayant plus de droits sur le système...

Pour les problèmes matériels, la stratégie la plus immédiate est de tester un matériel équivalent, cela permet de circonscrire le défaut [7]. Par exemple en empruntant un câble similaire, on sait tout de suite si le problème peut-être résolu facilement. Si cela ne change rien, il faut continuer les investigations.

Lorsque la led RGB ne clignote pas en bleu toutes les 4 secondes, cela traduit un problème matériel ou bien logiciel, notamment au niveau de l'UART [8]. Dans ce cas, il faut redémarrer la carte (*boot mode*). Deux modes de redémarrage sont accessibles :

— *Safe boot* : appui sur le bouton *reset* pour démarrer en mode standard, puis exécution du fichier `boot.py` suivie de l'exécution du fichier `main.py` (procédure standard) ;

— Il est possible de s'affranchir du mode précédent en connectant la broche P12 (ou G28 sur la carte d'extension) au 3.3V puis d'effectuer un *reset*. Dans ce mode, les fichiers `boot.py` et `main.py` ne sont pas exécutés.

Si le problème persiste, il est possible de recharger au préalable le firmware de la carte selon la procédure détaillée dans la section 7.4.3.

7.5 Première connexion au Wipy

Il est possible de vérifier le bon fonctionnement de la carte en passant en mode REPL (*Read–Eval–Print Loop*). Dans ce mode, la fonction `read` accepte une expression qui est décomposée en structure de données chargées en mémoire. Puis la fonction **`eval`** évalue l'expression et la fonction **`print`** affiche le résultat de l'évaluation. Enfin le processus retourne à la fonction `read`, créant ainsi une boucle (*loop*).

Afin de tester le mode REPL, il faut se connecter à la carte WiPy en ouvrant une session. Pour cela, il faut ouvrir une communication via le port USB en utilisant un terminal série.

La procédure est décrite ci-dessous selon le système utilisé (Windows/MacOs).

7.5.1 Approche Windows

Sous Windows, il faut utiliser un logiciel permettant d'établir ce type liaison, on peut utiliser par exemple Putty.

En sélectionnant la communication série (*Serial*) (cf. 7.6), il faut paramétrer :

— Le port *com* ouvert lors de la connexion de la carte sur un port USB (ici COM9) ;

— La vitesse de communication : 115200.

7. Cette technique est très utile tout au long de la construction du robot : quand quelque chose ne fonctionne pas comme prévu, on peut écarter rapidement le défaut matériel en testant avec un matériel équivalent. Il y a quand même un inconvénient : si la situation a été créée par une fausse manipulation, on risque de détruire le deuxième exemplaire du matériel testé. De plus, il n'est pas impossible que plusieurs matériels soient défectueux...

8. UART : *Universal Asynchronous Receiver Transmitter*

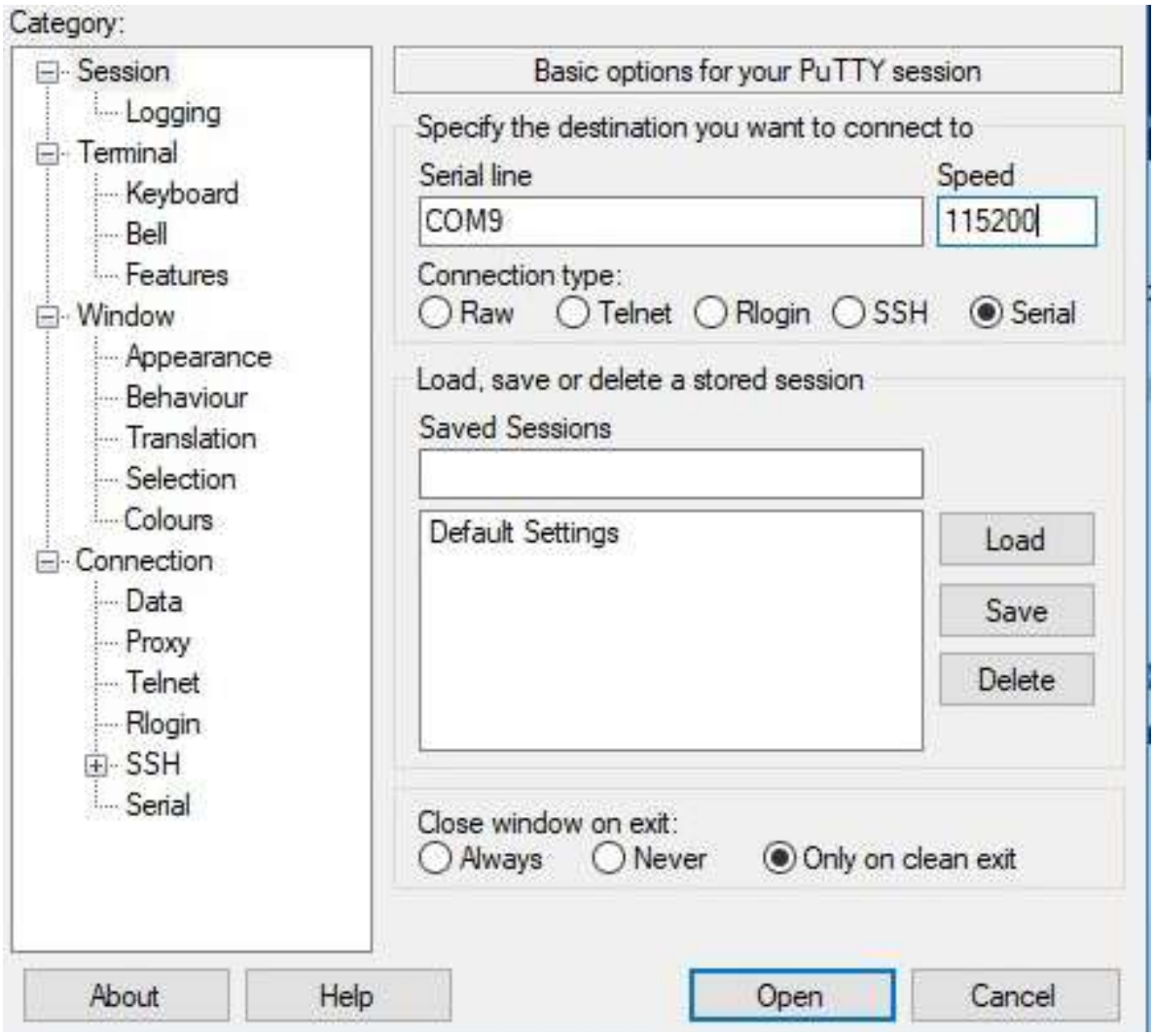


FIGURE 7.6 – Paramétrage du terminal série avec Putty

FIGURE 7.7 – Terminal série ouvert

L'ouverture de la session se traduit par la fenêtre ci-dessous (cf. 7.7). L'appui sur la touche «retour chariot» permet de passer en mode REPL, symbolisé par >>>. Il est alors possible d'exécuter sur la carte des instructions Python telle que le **`print`** (cf. 7.8).

7.5.2 Approche MacOs/linux

Pour se connecter au Wipy :

1. brancher le cable USB
2. chercher l'identifiant de la liaison usb-série (commande à taper dans une fenêtre Terminal) :

```
>ls /dev/tty.usb*
/dev/tty.usbserial-DQ00E50U
```

3. utiliser le programme telnet/screen pour se connecter :

```
>screen /dev/tty.usbserial-DQ00E50U 115200
```

Remarques :

— 115200 correspond à la vitesse de communication à utiliser

— le programme screen permet de gérer simultanément plusieurs "fenêtres" (en mode texte). Pour interargir avec screen il faut tapper Ctrl-a puis une autre touche :

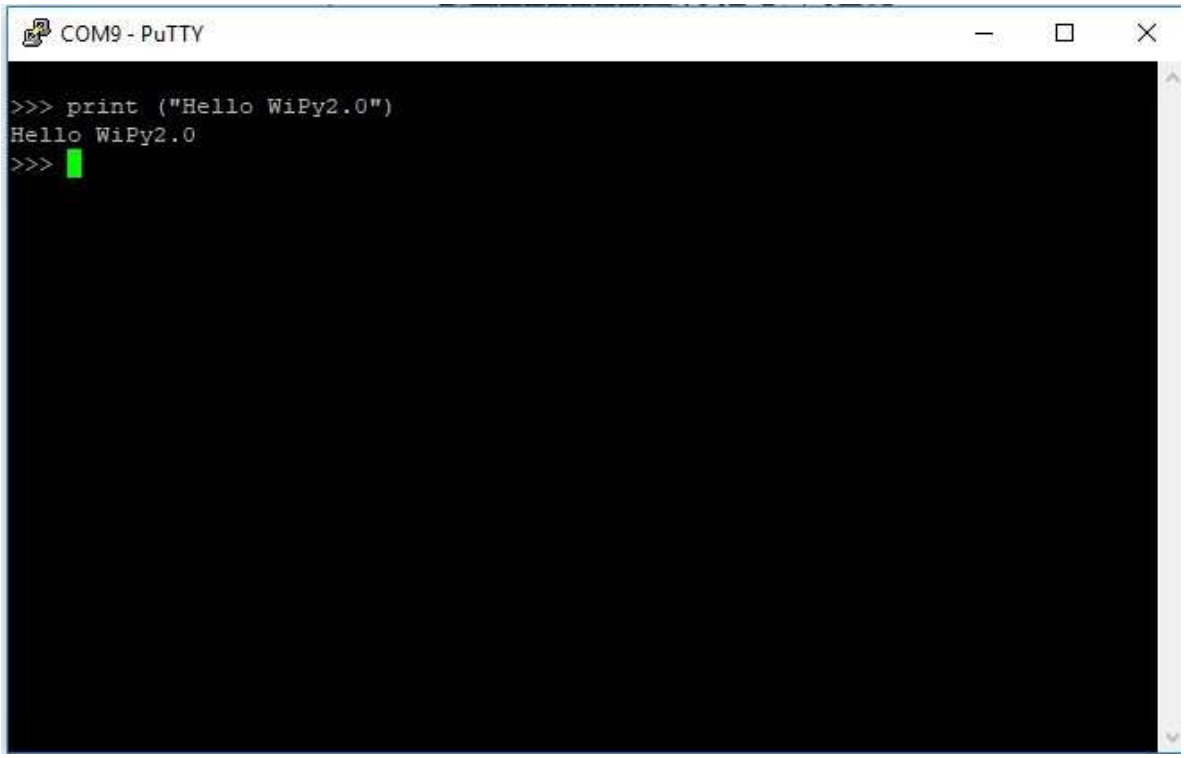


FIGURE 7.8 – Session REPL sous Putty : l'instruction **`print`** est exécutée.

— Ctrl-a-" pour avoir la liste des fenetres ouvertes
— Ctrl-a- ? pour avoir la liste des commandes possibles
— ... (il y a beaucoup de commandes, ce n'est pas toujourq très simple !)

On peut aussi se connecter en wifi :

1. le ssid commence par `wipy-wlan`
2. le mot de passe wifi est : `www.pycom.io` (attention la doc micropython indique autre chose : `www.wipy.io`)
3. cela permet de se connecter au Wifi du Pycom, ensuite pour accéder aux services, login : `micro` and mot de passe : `python`

7.6 Deuxième connexion au Wipy

La deuxième méthode que nous proposons utilise un éditeur de texte qui prend en charge la communication avec le Wipy. Si tout va bien, c'est plus simple.

1. Il faut installer l'outil Atom (`https://atom.io/`).
2. Dans Atom, il faut installer le module (*package*) permettant d'interagir avec le Wipy : `pymakr`. Si vous lisez la description de ce module, vous verrez qu'il permet d'intégrer une console REPL dans l'éditeur Atom ainsi de synchroniser des fichiers entre l'ordinateur et le wipy. La figure 7.9 montre la page permettant de sélectionner le module `pymakr`.
3. il faut indiquer au module `pymakr` l'adresse du port de communication : on peut obtenir cette information en cliquant sur le menu *More* (en bas à droite de la fenêtre de Atom), puis en choisissant la commande *Get serial ports*. La figure 7.10 montre le résultat sur MacOs.

Une fois ces étapes validées, l'éditeur Atom est opérationnel : cela signifie que l'on peut écrire des programmes (en python) et les transférer dans le Wipy.

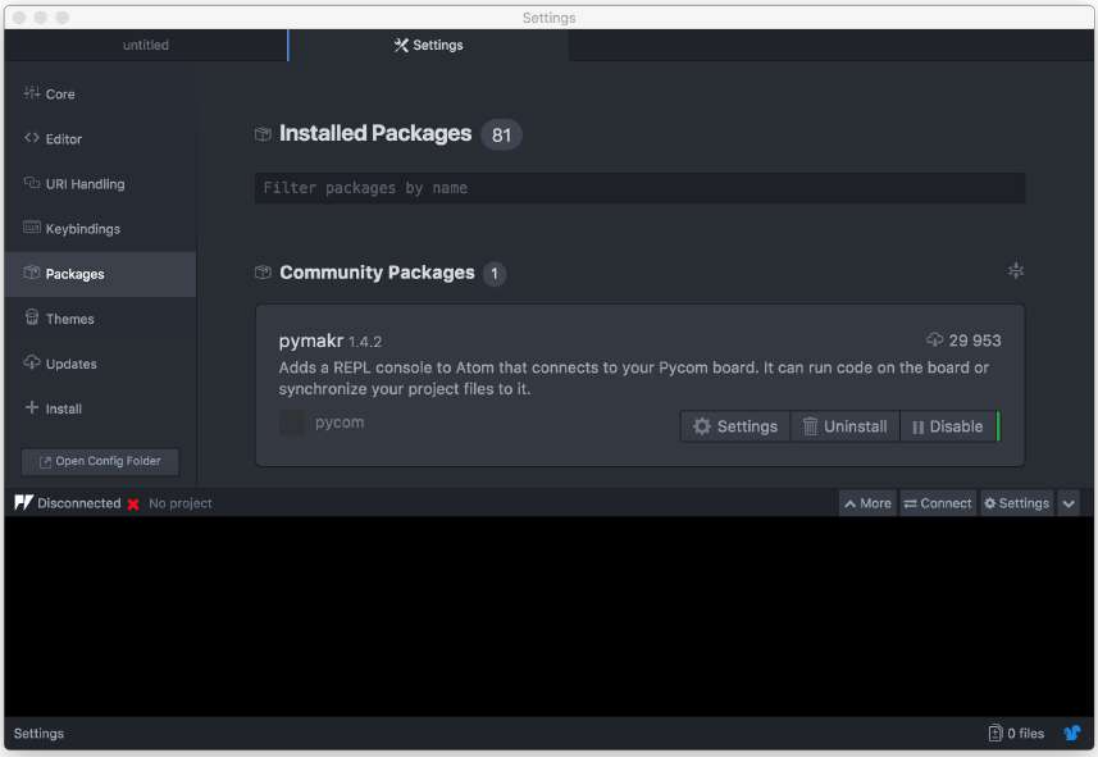


FIGURE 7.9 – Page *settings* de Atom permettant d'installer le module `pymakr` (les numéros de version peuvent différer).

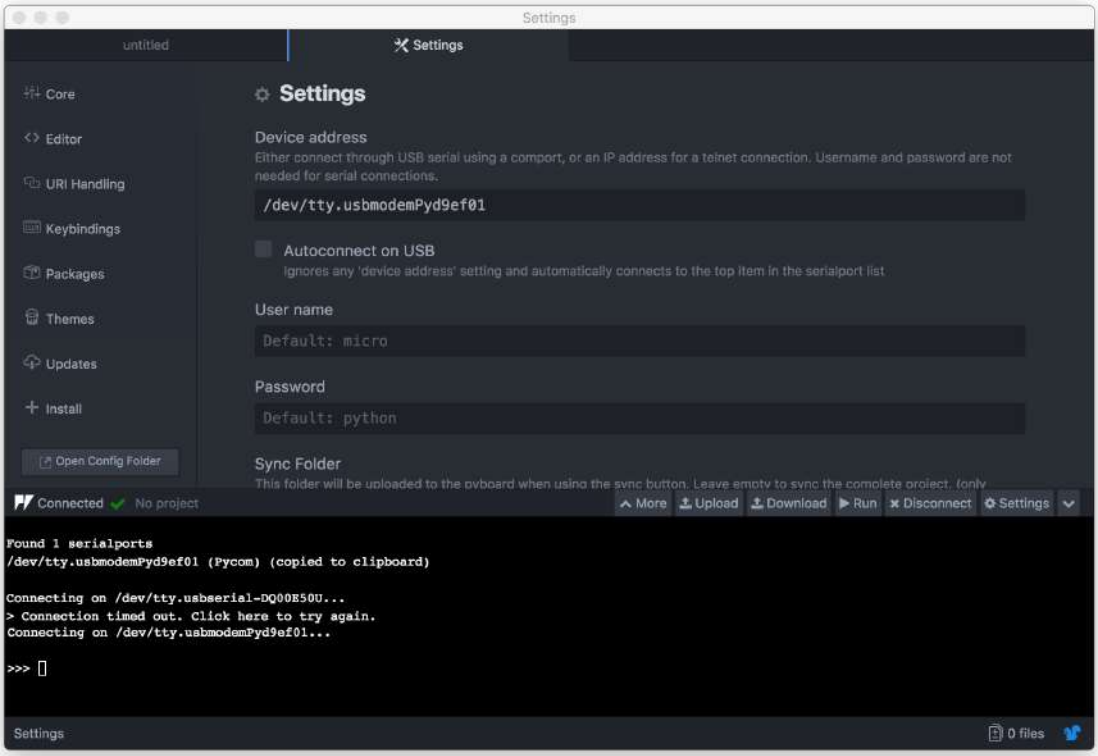


FIGURE 7.10 – Page *settings* de Atom permettant de configurer le module `pymakr`. La commande *Get serial ports* a permis d'obtenir l'information qui a été recopiée dans la section *Device address*. Cela a ensuite permis la connexion (la première tentative était un échec car l'adresse était mal configurée). La dernière ligne montre que la connexion est active et que les commandes peuvent être entrées (c'est ce que l'on appelle le mode console).

Chapitre 8

Quelques bases en python (micro-python)

(chapitre en phase de rédaction pour expliquer la programmation Python)

Le langage Python a été choisi parce qu'il peut être pris en main facilement et qu'il est de plus en plus enseigné en collège/lycée. Ce chapitre donne quelques bases mais il existe de nombreuses ressources en ligne permettant d'apprendre ce langage. Nous ne présentons qu'une petite partie de ce que ce langage permet de faire. En effet, nous nous concentrons sur ce qu'il nous est nécessaire de connaître pour programmer le robot.

Plus concrètement, nous allons utiliser une version « allégée » adaptée aux micro-contrôleurs : micropython (`https://micropython.org/`).

Exercice En guise d'introduction (en attendant que la documentation soit complète), lire les pages suivantes :

- `https://docs.pycom.io/gettingstarted/programming`
- `https://docs.pycom.io/gettingstarted/programming/micropython`
- `https://docs.pycom.io/gettingstarted/programming/examples`

8.1 Introduction

8.1.1 Les données

Les nombres

```
a = 3.14
b = 24
c = a*b
print(a)
print(b)
print(c)
```

Les chaînes de caractères

```
a = "un"
b = "mot"
c = "de␣plus"
print(a)
print(b)
print(c)
```

Exercice : Créer une variable `d` qui contiendra la concaténation[1] des variables `a`, `b` et `c`. Afficher la variable `d`.

Les listes

```
L=[]
L.append(5)
print(L)
L.append(21)
L.append('a')
print(L[1])
```

8.2 Structures répétitives : les boucles

Les boucles sont définies avec les mots-clé **`for`** et **`while`**.

```
for i in range(1,5):
    print(str(i))
```

Attention en python, les espaces en début de ligne sont IMPORTANTES : ils permettent de spécifier la portée d'action de la commande précédente.

Prenons un exemple (les espaces sont mises en valeur : on ne peut pas écrire ce symbole dans le code) :

```
for␣i␣in␣range(1,5):
␣␣␣␣print(str(i))
␣␣␣␣print(str(i**2))
```

dans ce cas, pour chaque valeur de i, il y a 2 affichages : i et i^2. Il ne se passera pas la même chose avec le code suivant :

```
for␣i␣in␣range(1,5):
␣␣␣␣print(str(i))
print(str(i**2))
```

Par la suite, nous n'indiquerons plus ces espaces aussi explicitement.

1. Utiliser l'opérateur +.

8.3 Les fonctions

Nous venons de voir que la boucle permet d'exécuter le même code plusieurs fois de suite. Cela permet d'éviter de recopier le même code autant de fois que l'on veut l'exécuter, ce qui devient compliqué lorsqu'on ne connaît pas le nombre d'exécutions à prévoir au moment où l'on écrit le programme. Il peut être assez utile de regrouper du code qui sera exécuté plusieurs fois sous la forme d'une fonction. Il est alors possible d'appeler le même code à différents emplacements du programme.

```
def rep_print():
    "fonction␣qui␣imprime␣1,␣2,...5␣quand␣on␣l'appelle"
    for i in range(1,5):
        print(str(i))
```

```
def rep_print(nb_repetitions):
    "fonction␣qui␣imprime␣1,␣2,...5␣quand␣on␣l'appelle"
    for i in range(1,nb_repetitions):
        print(str(i))
```

8.4 Les classes

Les classes sont un concept de programmation qui fait partie de la « programmation orientée objet« . Nous n'allons donc pas détailler ces concepts qui dépasseraient le cadre de ce guide. Cependant, il est utilisé dans de nombreuses occasions, et sans rentrer dans les détails, ce n'est pas très compliqué.

Une classe permet de définir un type de donnée beaucoup plus élaboré que les types classiques tel que les nombres, les chaînes de caractères... Cela permet de mettre sous un même nom, les données stockées et les traitements de ces données.

Prenons un exemple, le code suivant permet de proposer une classe qui permet de créer en mémoire une zone de données liée à un capteur (imaginaire)

```
class CapteurTemp:
    def __init__ (sef, temp_init = 0):
        self.temperature = temp_init

    def lire_temperature():
        self.temperature = LITCAPTEUR()
        return self.temperature
```

Cette classe `CapteurTemp` se compose de deux fonctions :

- `__init__` : cette fonction est appelée à la création d'une variable du type de la classe. Ici, la variable `self.temperature` (qui est interne à la classe : on le reconnaît à l'usage du prefixe `self`) est initialisée avec la valeur du paramètre `temp_init`.
- `lire_temperature` : cette fonction lit la température perçue par le capteur (les détails ne sont pas précisés ici), stocke la valeur dans la variable interne `self.temperature` et renvoie la valeur lue à celui qui a appelé la fonction.

Voici comment s'utilise cette classe : on peut créer plusieurs variables permettant de représenter plusieurs capteurs (une variable par capteur) :

```
capt1 = CapteurTemp(5) #la temp initiale vaut 5
capt2 = CapteurTemp(2) #la temp initiale vaut 2

#appel de la fonction lire_temperature
#pour les 2 capteurs
temp1 = capt1.lire_temperature()
temp2 = capt2.lire_temperature()

#affichage
print(temp1)
print(temp2)
```

Ce qu'il faut remarquer :

— la création d'une variable ressemble à un appel de fonction : `capt1 = CapteurTemp()`

— l'appel d'une fonction faisant partie de la classe utilise un point entre le nom de la variable et la fonction appelée : `capt1.lire_temperature()`

Chapitre 9

Premiers tests

Une fois la carte Wipy installée sur la carte d'extension, l'environnement logiciel configuré, nous allons procéder à l'écriture de nos premiers programmes. Dans le domaine de la programmation avec des interactions informatiques/électronique, l'exercice de base consiste à faire clignoter une led. On commence toujours par là !

Ce premier programme permet aussi de mettre en place le cycle écriture du code-transfert sur Wipy.

9.1 Télécharger un programme

L'interface Atom/pymakr permet de lancer du code micro python sur le Wipy en appuyant sur le bouton *Run*. Cette technique est rapide pour tester le bon fonctionnement du code mais elle présente deux inconvénients : le code n'est pas enregistré sur le Wipy (il y est juste exécuté) et ce code ne peut faire appel qu'aux fonctions standards (qui sont pré-enregistrées). Pour contourner ces deux problèmes, le code et les fonctions supplémentaires (appelées librairies) peuvent être enregistrés sur le Wipy : *Upload*. Plus précisément, le bouton *Upload* enregistre tout ce qui est présent dans le dossier du projet en cours.

La démarche pour créer un programme sera la suivante :

1. créer un dossier qui contiendra les programmes, librairies ;
2. écriture d'un fichier contenant le code micropython dans le dossier : ce fichier doit s'appeler `main.py`
3. transfert du projet dans le Wipy : bouton *Upload*

Exercice :

1. créer un dossier `test0`
2. ouvrez ce projet dans Atom (commande *Open Folder...* du menu *File*)
3. créez un nouveau fichier vide (commande *New File* du menu *File*), en lui donnant le nom `main.py` (ce fichier doit être dans le dossier `test0`)
4. tapez le code suivant dans le nouveau fichier vide :

```
for i in range(10):
    print("tout␣fonctionne␣("+str(i)+")")
```

5. sauvegarder le fichier (commande *Save* du menu *File*)
6. transférer le projet sur Wipy : commande *Upload*
7. lancer le programme : commande *Run* (peut-être inutile car le programme peut démarrer automatiquement)
8. expliquez les 2 lignes de programme

9.2 Test de la LED

Ce programme permet de faire clignoter une led présente sur la carte d'extension. Cela permet de comprendre comment on interagit avec une ressource matérielle (simple) ainsi que la gestion du temps (pour clignoter). De plus, la structure générale du programme sera similaire par la suite. Donc en plus de vérifier que tout fonctionne, ce premier programme donne un modèle pour tous les programmes qui vont suivre. Il est donc important de comprendre tous les détails !

9.2.1 Étapes

Les programmes que nous produirons auront tous la même structure en 2 phases :

1. initialisations (variables, entrées-sorties...)
2. boucle infinie (perception, analyse, action,...)

Voici les principales étapes de ces 2 phases :

1. Initialisation
   (a) inclusion des bibliothèques
   (b) déclaration des variables
   (c) définition des fonctions
2. Boucle infinie :
   (a) lecture des capteurs
   (b) action
   (c) temporisation

L'exercice précédent a permis de vérifier que le transfert fonctionne. Nous allons maintenant rentrer dans le détail du programme permettant de faire clignoter une Led.

9.2.2 Principales instructions

Inclusion des bibliothèques

Pour utiliser les ressources matérielles, en particulier les broches (*Pin*), il faut inclure la bibliothèque (aussi appelée librairie) `machine`. En python, cela s'écrit de la façon suivante :

```
from machine import Pin
```

Pour gérer le temps (durées, attendre...), la bibliothèque se nomme `time` :

```
import time
```

Notez que ces deux écritures permettent d'inclure une bibliothèque dans le programme : cela signifie que l'on peut ensuite faire appel à des fonctions qui sont prévues dans ces bibliothèques. La présentation différente de ces instructions (qui ont donc le même but) ont des conséquences :

— la première : `from ... import` permet de ne retenir que l'élément `Pin` de la bibliothèque `machine` (il y a d'autres éléments dans cette librairie mais nous n'allons pas y faire appel). Cette façon d'inclure une partie de la librairie `machine` permet d'appeler, dans toute la suite du programme, directement `Pin` (sans avoir à repréciser que cela fait partie de `machine`) ;

— la deuxième : `import ...` permet d'inclure toute la librairie `time`. Cependant, il faudra préciser que les éléments utilisés dans cette librairie y sont attachés.

Déclaration des variables

Une variable `LedP9` va être créée pour accéder à l'allumage de la led sur la broche P9. Comme nous utiliserons cette led pour transmettre une information (éclairement/éteinte), cette broche sera configurée en sortie (`mode=Pin.OUT`).

```
LedP9 = Pin('P9', mode=Pin.OUT)
```

La variable `LedP9` sera ensuite utilisée pour faire en sorte qu'au démarrage, la led ne soit pas activée (valeur 0) :

```
LedP9.value(0)
```

Enfin, avant de passer au cycle de répétition, un quart de seconde est laissé afin que le changement soit visible :

```
time.sleep(0.25)
```

Cette instruction `sleep()` vient de la librairie `time` incluse plus haut. On remarque que la bibliothèque ayant été incluse par la forme `import ...`, il est nécessaire, pour atteindre l'instruction, d'écrire `time.sleep()`.

Cycle principal

Une fois les initialisations terminées, les variables et données nécessaires sont prêtes. Le microcontrôleur rentre alors dans un cycle (ou une boucle) qui ne s'arrêtera que lorsque qu'il ne sera plus alimenté (on peut aussi appuyer sur le bouton *Run* tant que l'ordinateur est connecté). Pour obtenir ce comportement, une boucle qui ne s'arrête jamais, on peut utiliser l'instruction `while` en lui donnant une condition d'arrêt qui ne sera jamais fausse. Par exemple, on peut utiliser `True` qui, sans surprise, prend toujours la valeur Vrai :

```
while True:
    #suite
```

En écrivant des instructions à la place du commentaire `#suite`, celles-ci seront répétées indéfiniment.

Pour faire clignoter une led, il faut l'allumer, attendre, l'éteindre, attendre, l'allumer, attendre, l'éteindre,...

Cela peut se faire de la façon suivante (il n'y a rien de nouveau dans ce code) :

```
  Led_P9.value(0)
  time.sleep(0.5)
  Led_P9.value(1)
  time.sleep(0.5)
```

Il faut ajouter ce code « dans » la boucle infinie. Cela signifie que les instructions précédentes doivent être décalées d'une même marge par rapport à l'instruction `while True:`

Cela donne :

```
while True :
  Led_P9.value(0)
  time.sleep(0.5)
  Led_P9.value(1)
  time.sleep(0.5)
```

Tout y est, le programme complet (avec quelques commentaires et affichages supplémentaires) est donné ci-dessous :

```
# main.py
# Programme LedP9_clignote : piloter la led de la carte
# d'extension et la faire clignoter
# Led carte d'extension sur le port P9
# Cavalier de la carte d'extension branche

from machine import Pin
import time

# initialize ``P9`` in gpio mode and make it an output
Led_P9 = Pin('P9', mode=Pin.OUT)
Led_P9.value(0)
time.sleep(0.25) # temporisation de 0.25s
while True :
  Led_P9.value(0)
  print("Led_P9 OFF")
  time.sleep(0.5) # temporisation de 0.5s
  Led_P9.value(1)
  print("Led_P9 ON")
  time.sleep(0.75) # temporisation de 0.75s
```

Exercice :

1. tester le programme ;
2. modifier le programme pour que la led soit allumée $1,5$ secondes toutes les $3,6$ secondes ;
3. modifier le programme pour que la led clignote de plus en plus vite.

9.2.3 Conclusion

Pour aller plus loin, il faut plonger dans les détails. Par exemple, la classe `Pin` de la librairie `machine` mériterait plus d'attention : lisez `https://docs.pycom.io/firmwareapi/pycom/machine/pin`!

9.3 Plus de précisions sur le temps

Nous avons vu que la librairie `time` permettait de mettre le programme en attente en s'endormant (fonction `sleep()`).

Nous aurons besoin de mesurer le temps qui passe. Cela est possible grâce à un composant et une librairie associée : `RTC` (pour *Real Time Clock*).

Exercice : En vous appuyant sur la description de la librairie `RTC` (`https://docs.pycom.io/firmwareapi/pycom/machine/rtc`), écrire un programme qui écrit les date et heure courantes toutes les 5 secondes. On veut que l'affichage soit identique à ce qui suit :
`5/10/2018  15:42:20.55808`
(jour/mois/année heures:minutes:secondes.microsecondes)

Chapitre 10

Comment capter la température et sauvegarder ses données sur la carte SD

Dans ce chapitre, nous allons prendre en main la communication entre le Wipy et le capteur environnemental. Cette communication passe par un bus :

> Un bus informatique est un dispositif de transmission de données partagé entre plusieurs composants d'un système numérique.[...]. Le bus informatique est la réunion des parties matérielles et immatérielles qui permet la transmission de données entre les composants participants. [1]

Ce chapitre est aussi l'occasion de vous familiariser avec l'utilisation d'une bibliothèque.

10.1 Le bus I2C

I2C signifie en anglais *Inter-Integrated Circuit*. Il y a à la fois un aspect matériel (les fils, l'électronique) et un aspect logiciel (protocole de communication implémenté par des fonctions dans des bibliothèques).

Pour établir une connexion I2C, il faut 2 fils (et une masse), dont les noms sont :

— SDA (Serial Data Line) : ligne de données bidirectionnelle,

— SCL (Serial Clock Line) : ligne d'horloge de synchronisation bidirectionnelle.

Chaque ligne peut transmettre un niveau haut ou bas électrique, ce qui se traduit :

— la donnée 1 ou 0 sur la ligne SDA

— une impulsion d'horloge sur la ligne SCL

La valeur de la ligne SDA est lue pendant l'impulsion d'horloge (c'est-à-dire lorsque la ligne SCL est au niveau haut (voir figure 10.1, `https://fr.wikipedia.org/wiki/I2C`).

1. `https://fr.wikipedia.org/wiki/Bus_informatique`

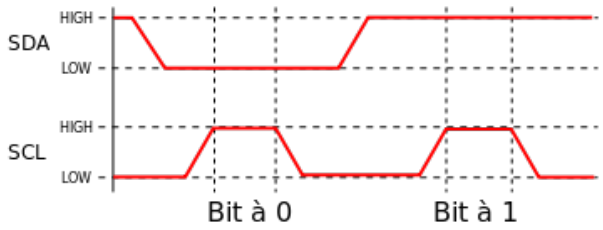


FIGURE 10.1 – Principe de codage des bits sur un bus I2C (wikipedia).

10.2 Le capteur BME280

10.2.1 Présentation générale

Le capteur BME280 (fabriqué par BOSCH) est un capteur permettant de mesurer la température, l'humidité et la pression atmosphérique.

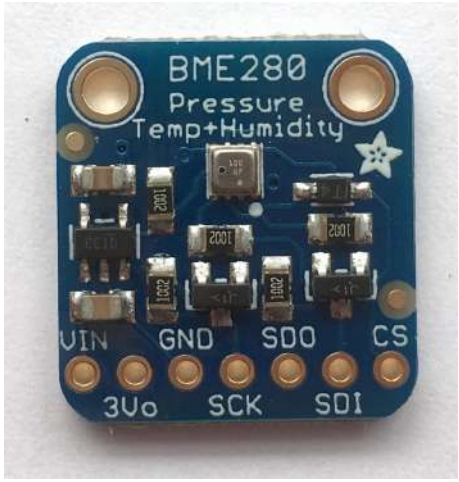


FIGURE 10.2 – Carte contenant le capteur BME280.

Pour tous les détails du capteur, il faut toujours mieux se référer à sa fiche technique (*datasheet*) : `https://www.bosch-sensortec.com/bst/products/all_products/bme280` mais les 54 pages ne sont pas vraiment d'une lecture facile. Pour cette raison, nous allons les présenter plus simplement et nous concentrer sur ce qui nous intéresse dans le cadre du robot.

Tout d'abord, nous n'allons pas utiliser le capteur tel qu'il est fourni par BOSH. Sa taille de 2,5 mm de côté est un avantage car il est vraiment petit, mais cela complique la soudure de ce type de composant sur une carte. C'est pour cette raison (entre autres) que nous utiliserons une carte sur laquelle le composant est déjà fixé. Cette carte est commercialisée par Adafruit : `https://www.adafruit.com/product/2652`. Comme le BME280 est le composant le plus important de cette carte, nous parlerons indifféremment des deux (le composant et la carte) sous le nom BME280 (voir figure 10.2).

Exercice : Repérer le composant BME280 sur la carte Adafruit.

10.2.2 câblage

Les 7 broches sont étiquetées de la façon suivante :
- `Vin` : alimentation de la carte. La carte accepte des tensions comprises entre 3V et 5V grâce à la présence d'un régulateur (qui régule à 3.3V pour le capteur) ;
- `3Vo` : sortie régulée à 3.3V (pour profiter du travail du régulateur)
- `GND` : masse
- `SCK` : horloge I2C
- `SDO` : non utilisée en I2C
- `SDI` : données I2C
- `CS` : non utilisée en I2C

Les broches `SDO` et `CS` sont utilisées lorsque le capteur utilise un autre protocole de communication : SPI. Nous ne détaillons pas cet aspect par la suite.

Donc pour utiliser le capteur avec le protocole de communication I2C, nous avons besoin de connecter les broches `Vin`, `GND`, `SCK` et `SDI`.

Attention : les noms des broches sur le capteur ne correspondent pas exactement aux noms utilisés sur le Wipy... `SDI` sur le capteur correspond à `SDA` sur Wipy et `SCK` sur le capteur correspond à `SCL` sur le Wipy. Ce n'est pas pratique, mais c'est comme cela !

Selon que l'on dispose d'une carte de connexion (voir figure 10.3) ou non, l'un ou l'autre des deux exercices suivants sera faisable.

Exercice : Branchement du capteur directement sur la carte d'extension Pycom

1. repérer les broches de la carte d'extension qui permettront de brancher le capteur :

capteur BME280	carte Wipy
`Vin`	P...
`GND`	P...
`SCK`	P...
`SDI`	P...

2. effectuer les branchements : la masse est toujours connectée par un fil noir. Il n'y a pas de norme pour les autres connexions, si ce n'est qu'il faut choisir des fils de couleurs différentes pour faciliter le repérage.
3. faire vérifier son branchement.

Exercice : Utilisation de la carte de connexion

1. Il suffit d'insérer la carte BME280 dans le connecteur (voir figures 10.4, 10.3 et 10.5).

10.3 Mise en œuvre

10.3.1 Installation de la bibliothèque BME280

Une bibliothèque (ou *library*, en anglais) est une collection de fonctions, constantes, variables...associées à une fonctionnalité particulière. Dans notre cas, lorsque nous parlons de la

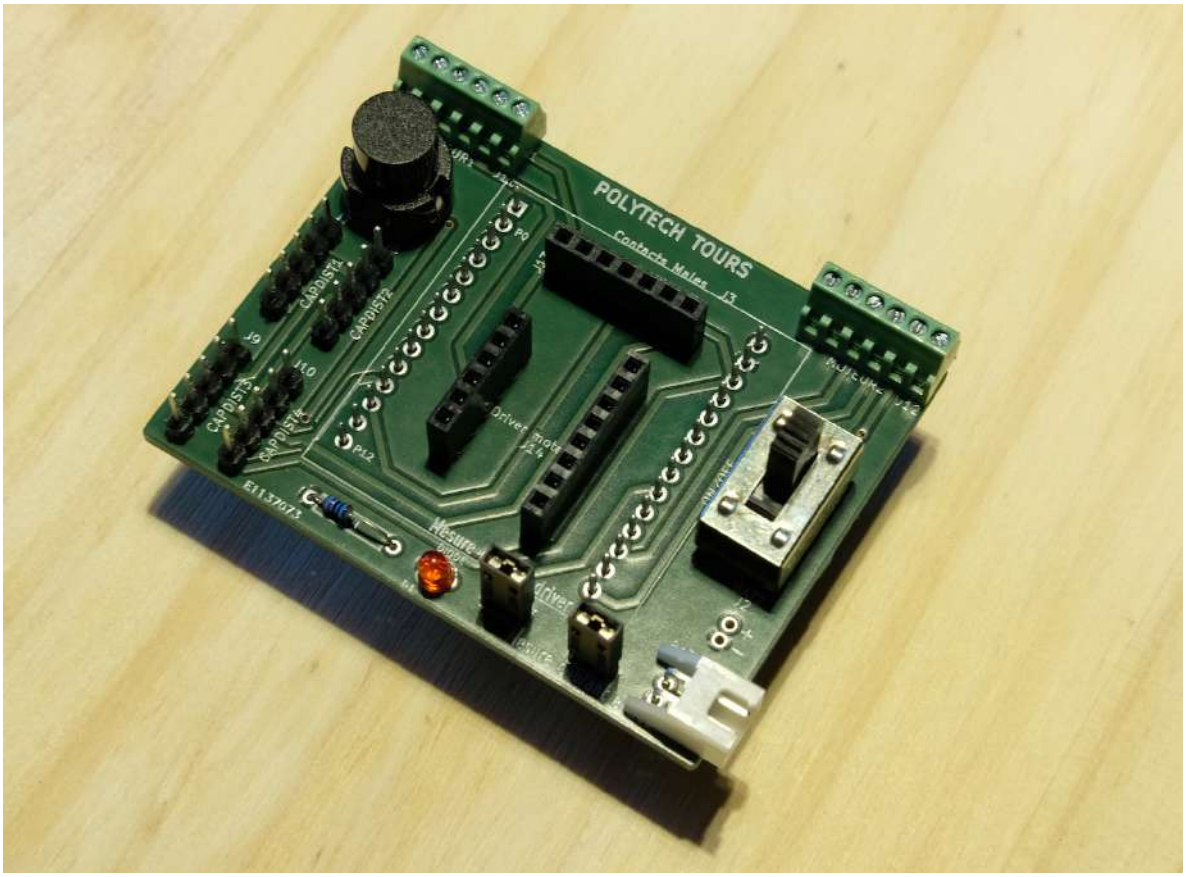


FIGURE 10.3 – Carte de connexion.

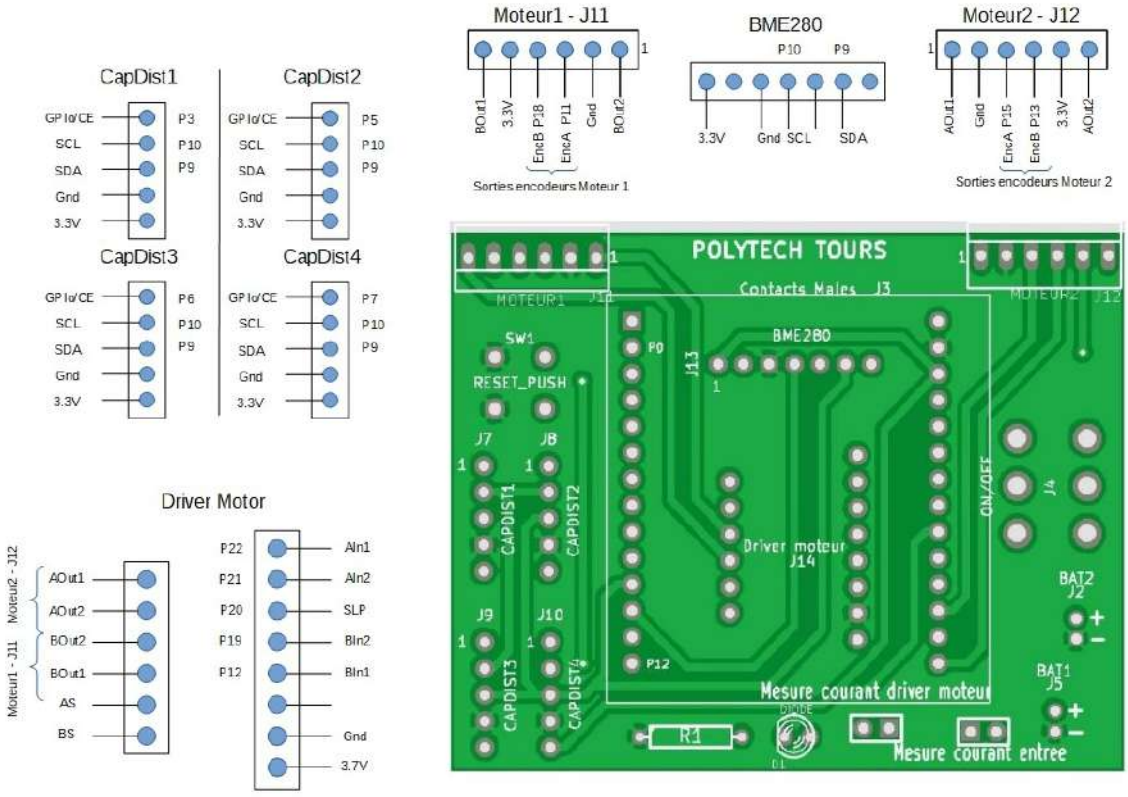


FIGURE 10.4 – Plan de la carte de connexion.

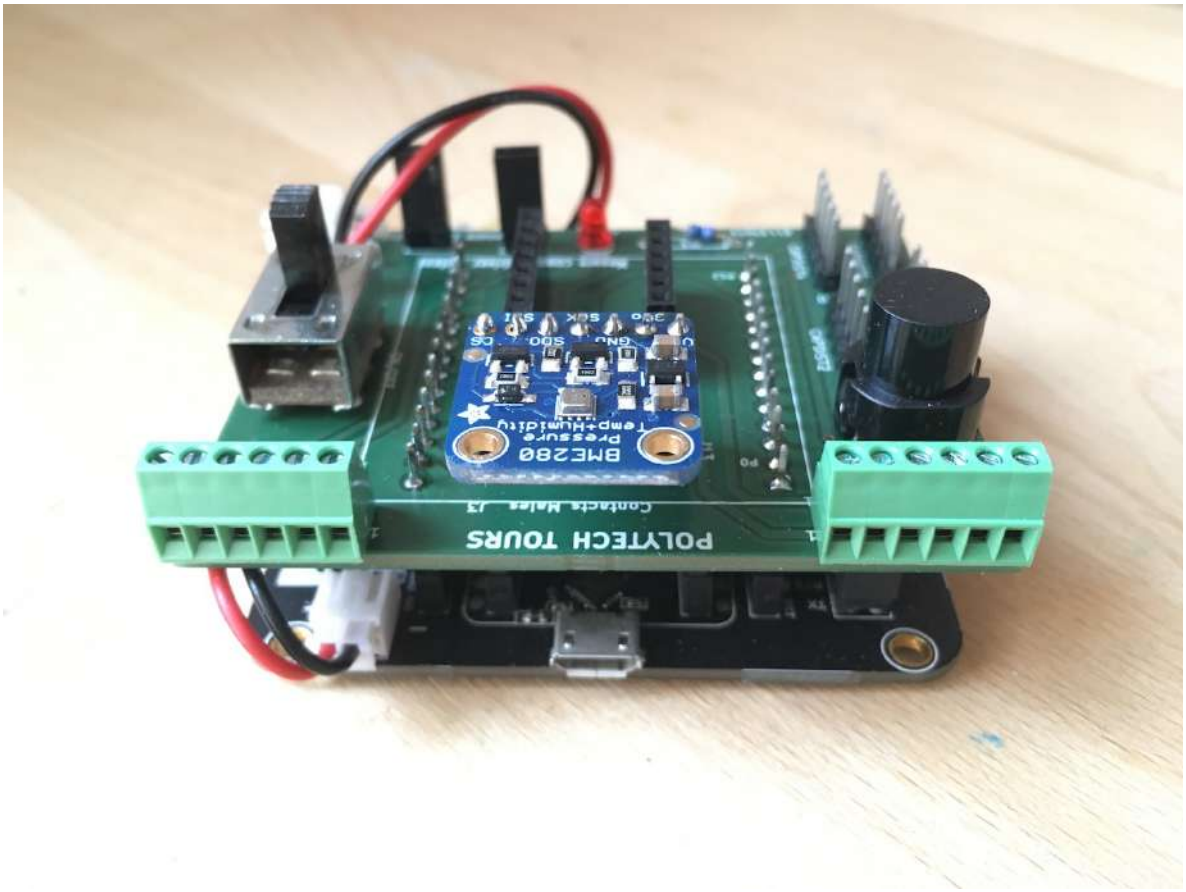


FIGURE 10.5 – Installation du capteur BME280 sur la carte de connexion.

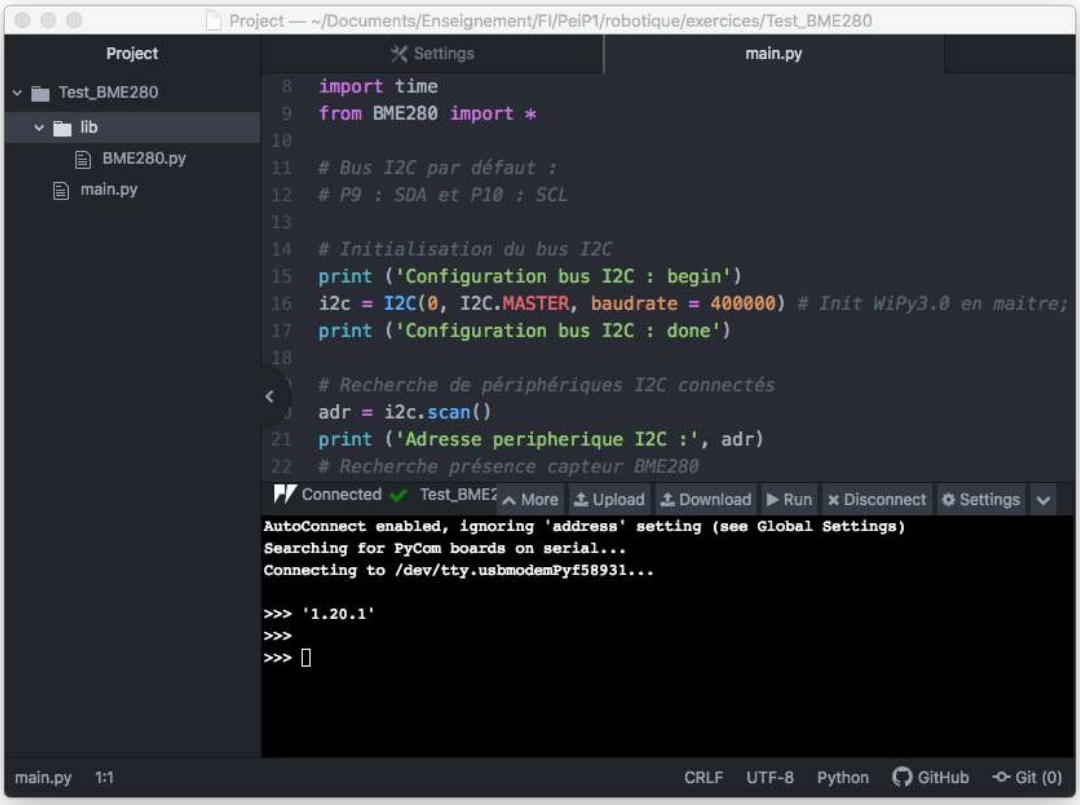


FIGURE 10.6 – Fenetre principale Atom : l'onglet « Project » doit se présenter de cette façon pour que l'opération *upload* se déroule correctement.

bibliothèque BME280, il s'agit d'un fichier contenant du code python qui permet d'interagir avec le circuit qui accède au BME280.

Pour utiliser une bibliothèque, il faut créer un dossier `lib` dans le dossier de son projet et y copier le fichier de la bibliothèque. Ainsi, lorsque le code python du fichier `main.py` fait appel à une fonction proposée par une bibliothèque, le code à exécuter sera cherché dans les fichiers qui ont été copiés dans `lib`.

Attention, dans Atom, il faut que le fichier `main.py` et le dossier `lib` soit dans le même dossier. Il faut que la fenêtre *Project* ne contiennent rien d'autre, sinon tout est copié sur le Wipy (au moment de l'*upload*) et cela ne fonctionne pas... La figure 10.6 montre ce que vous devez avoir pour chaque projet (partie gauche).

Ensuite dans le programme, pour avoir accès aux fonctions de la bibliothèque, il suffit d'écrire au début du fichier `main.py` :

```
from BME280 import *
```

10.3.2 Mise en place de la connexion I2C

La bibliothèque BME280 fait elle-même appel à la librairie `machine` qui contient la définition de la classe I2C : la définition d'une classe est expliquée dans la section 8.4. La documentation de la prise en charge de l'I2C sur Wipy est disponible ici : `https://docs.pycom.io/firmwareapi/pycom/machine/i2c`.

La classe I2C permet de créer une variable représentant le bus I2C :

```
bus_i2c = I2C(...)
```

Attention, la classe est écrite en majuscules ! À la place des trois points, il faut indiquer les paramètres de la liaison : en particulier le numéro du bus (la valeur 0 convient). On peut mettre d'autres paramètres (si les branchements ne sont pas standards).

Une fois la variable `bus_i2c` créée, on peut initialiser les paramètres de la liaison :

```
bus_i2c.init(...)
```

Cette fonction prend trois paramètres : les broches de gestion du bus i2c (valeur 0 : broches par défaut), le type de connexion (`I2C.MASTER`) et la vitesse (`baudrate = 400000`). En python les paramètres sont séparés par des virgules et peuvent être nommés (cela signifie que l'on donne le nom du paramètre à qui on donne une valeur, c'est le cas ici du paramètre qui s'appelle `baudrate`). Donc, l'appel à la fonction `init()` qui est liée à la variable `bus_i2c` se fait de la façon suivante :

```
bus_i2c.init(0,I2C.MASTER, baudrate = 400000)
```

La fonction `scan()` renvoie les adresses des périphériques connectés au bus I2C :

```
adr = bus_i2c.scan()
print ('Adresse␣peripherique␣I2C␣:', adr)
```

Exercice

1. créer un nouveau projet/dossier, `test_I2C`
2. dans ce projet, créer un fichier `main.py` et écrire un programme qui affiche l'adresse I2C du périphérique (dans notre cas il s'agit du BME280).

Remarques Si cela ne fonctionne pas...

— il faut inclure la bibliothèque `machine` pour avoir accès à la classe `I2C` : `from machine import I2C`
— l'adresse I2C du capteur BME280 vaut 119 (pour la carte Adafruit, cela peut différer pour d'autres cartes)
— il faut que le capteur soit bien branché : pas de capteur : pas de communication I2C...

10.3.3 Mise en place de la lecture du capteur BME280

La première étape est d'inclure la bibliothèque BME280 dans son projet. Il faudra répéter cette étape pour tous les projets qui utilisent ce capteur.

Exercice

1. créer un nouveau projet/dossier, `test_BME280`
2. dans le dossier de ce nouveau projet, ajouter un sous-dossier `lib`
3. copier le fichier `BME280.py` dans ce dossier (ce fichier est disponible sur l'ENT)
4. créer un fichier `main.py` et tester que l'inclusion de la bibliothèque fonctionne : vous pouvez reprendre le code de l'exercice précédent mais à la place de :
   ```
   from machine import I2C
   ```
   vous écrivez :
   ```
   from BME280 import *
   ```
   Si cela fonctionne : le résultat est identique.

Par la suite, les exercices vont permettre de compléter le même projet.

L'étape suivante consiste à lire sur le bus I2C l'identifiant du capteur BME280. Cela se fait par la ligne suivante :

```
Id_BME280 = bus_i2c.readfrom_mem(BME280_I2C_ADR,
    BME280_CHIP_ID_ADDR, 1)
```

Cette commande lit 1 octet de la mémoire du périphérique repéré par l'adresse : `BME280_I2C_ADR` qui est une constante (`0x77`) définie dans la bibliothèque BME280 pour correspondre à l'adresse I2C du capteur BME280 (119 en décimal pour le shield Adafruit BME280). La lecture commence à l'adresse `BME280_CHIP_ID_ADDR` (qui est aussi une constante définie dans la bibliothèque BM2280 : `0xD0`). Cette constante correspond à l'adresse du registre contenant l'identifiant du capteur.

Le retour de la fonction `readfrom_mem` est stocké dans la variable `Id_BME280`. Si on veut l'afficher (pour vérifier que tout va bien, que notre capteur est bien celui que l'on a branché...), on peut écrire :

```
print ('Valeur Id_BME280 : ', hex (Id_BME280[0]))
```

Remarque Vous avez probablement remarqué que les adresses sont souvent écrites sous la forme `0x??`, cela signifie qu'elles sont écrites en hexadécimal. L'intérêt de cette écriture est que la valeur d'un octet s'écrit avec 2 symboles : de `0x00` à `0xFF` (au lieu de 0 à 255 en décimal où 3 symboles sont nécessaires).

Exercice À partir de l'exercice précédent, compléter le code pour que l'on puisse vérifier, en l'affichant, l'identifiant du capteur.

Il reste une chose à faire avant de pouvoir lire des informations de température avec le capteur : il faut calibrer le capteur. Pour ce faire, nous allons créer une variable `capteur_BME280` qui représentera le capteur :

```
capteur_BME280 = BME280 (BME280_I2C_ADR,  bus_i2c)
```

Il est à noter que la bibliothèque cache à l'utilisateur une partie des opérations nécessaires à la calibration du capteur. Par exemple, les paramètres suivants permettent de configurer le capteur :

— `BME280_OVERSAMPLING_16X` : Valeur de sur-échantillonnage pression

- `BME280_OVERSAMPLING_16X` : Valeur de sur-échantillonnage température
- `BME280_OVERSAMPLING_16X` : Valeur de sur-échantillonnage humidité
- `BME280_FILTER_COEFF_2` : Coefficients filtre
- `BME280_STANDBY_TIME_125_MS` : Délai d'inactivité
- `BME280_NORMAL_MODE` : Choix du mode de fonctionnement du capteur (sommeil / force / normal)

L'utilisation du capteur nécessite de récupérer les paramètres de calibration « usine » par la commande :

```
capteur_BME280.Calibration_Param_Load()
```

Il ne reste qu'une chose à connaître : demander au capteur la lecture de la température, de l'humidité et la pression. Il y a une fonction disponible dans la bibliothèque pour chaque grandeur : `read_temp()`, `read_pression()` et `read_humidity()`.

Exercice À partir de l'exercice précédent (et des explications précédentes...) :

1. compléter le code pour afficher les trois mesures (température, pression et humidité) toutes les 5 secondes. Le résultat de l'affichage devrait ressembler à ça :

   ```
   >>>
   Adresse peripherique I2C : [119]
   Valeur Id_BME280 :  0x60
   temp = 28.06706
   pres = 1012.371
   humi = 36.28979
   ----------------
   temp = 28.06103
   pres = 1012.361
   humi = 36.33525
   ----------------
   temp = 28.04926
   pres = 1012.366
   humi = 36.34102
   ```

   Il es conseillé de mettre les grandeurs dans des variables, puis d'afficher ces variables.
2. déplacer le capteur (ou soufflez dessus) pour vérifier que la température et l'humidité varient.
3. rechercher sur internet la valeur de la pression atmosphérique, à l'endroit où vous vous trouvez.

10.4 Enregistrer les données sur une carte SD

Afin d'exploiter les informations environnementales perçues par les capteurs du robot, il faut les enregistrer. Nous pourrions les transmettre directement à un serveur prêt à les recevoir et à les stocker. Cependant, l'environnement réseau n'est pas toujours fiable, notamment lorsque le nombre de robots/capteurs est important. C'est alors plus prudent de prévoir une sauvegarde sur une carte SD embarquée sur le robot. Nous allons tester cette

fonctionnalité : heureusement, il y a une bibliothèque gérant les cartes SD pour le Wipy : `https://docs.pycom.io/firmwareapi/pycom/machine/sd`.

Sans carte d'extension, c'est-à-dire avec le Wipy tout seul, il faut utiliser certaines broches pour connecter un lecteur de carte SD (P4, P8 et P23) mais lorsque l'on utilise la carte d'extension, la connexion de ces broches est déjà faîte pour pouvoir utiliser le lecteur de carte présent sur la carte d'extension.

Il n'y a donc rien de spécial à faire au niveau branchement. Par contre, il faut savoir que les broches P4, P8 et P23 ne peuvent pas être utilisées à autre chose si on veut accéder à la carte SD.

Sans rentrer dans les détails, les fichiers sont stockés et organisés de façon différente selon les supports que l'on utilise (carte SD, disque dur, etc.). Afin de pouvoir accéder à des fichiers en lecture ou en écriture, il faut mettre en place le « système de fichiers » afin d'utiliser la bonne organisation selon le support. On parle de montage du système de fichier (*mount*).

Ensuite, pour lire ou écrire dans un fichier, il faut l'ouvrir (*open*) puis le fermer à la fin : (*close*).

L'exemple de code suivant résume les étapes :

```
from machine import SD
import os

sd = SD()
os.mount(sd, '/sd')

# ouverture en ecriture : 'w'
f = open('/sd/test.txt', 'w')
#ecriture d une chaine de caractere
f.write('Information de temp/hum/pression')
f.close()
```

Remarques

— on écrit des chaînes de caractères dans le fichier. Cela signifie que si l'on veut écrire un nombre, il faut le transformer en chaîne avec la fonction `str()` (*string* signifie chaîne en anglais) :

```
nombre = 312
f.write(str(nombre))
```

De plus, pour les nombres à virgule, on peut chercher à limiter le nombre de chiffres après la virgule. Par exemple, limitons à 2 :

```
nombre = 32.552643
f.write(str("%.2f"%nombre))
```

Exercice

1. créer un projet `test_sd` qui va contenir les fichiers sources
2. initialiser la carte SD et créer un fichier « info.csv ». Les fichiers au format CSV contiennent des données en colonnes séparées par un point-virgule. Écrire une première ligne dans le fichier avec les en-têtes de colonne :

`AA;MM;JJ;HH;MM;SS;temp;humi;pres\r\n`
(année, mois, jour, heures, minutes, secondes... le tout terminé par des caractères de retour à la ligne).

3. faire une boucle pour répéter 5 fois les opérations suivantes :
   — récupérer l'heure courante (voir l'exercice avec RTC, page 48)
   — choisir des valeurs pour la température, ...
   — écrire une ligne dans le fichier, contenant les informations de l'heure puis des informations de température etc. en les séparant par des points virgules
4. une fois le programme exécuté, extraire la carte SD et ouvrir le fichier CSV sur un ordinateur pour vérifier qu'il est au bon format.

10.5 Conclusion

Logiquement, l'exercice suivant consiste à assembler les exercices précédents :

Exercice Écrire un programme qui mesure toutes les 5 secondes la température, l'humidité et la pression et qui les enregistre dans un fichier CSV sur la carte SD. Les colonnes doivent être dans l'ordre :
`AA;MM;JJ;HH;MM;SS;temp;humi;pres`.

Chapitre 11

Comment faire bouger son robot

Ce chapitre est consacré à la commande des moteurs.

11.1 Introduction

Les moteurs qui ont été retenus sont des moteurs à courant continu, équipé d'encodeurs (DFRobot Micro Metal Gearmotor with Encoder FIT0483) (cf. figure 11.1). Cette technologie de moteur à courant continu a été retenue car les moteurs sont légers, ne prennent pas trop de place et consomment peu d'énergie. D'autres modèles plus puissants ou plus précis sont envisageables (servomoteurs ou moteurs pas-à-pas) mais cela sera au détriment de l'encombrement ou de la consommation électrique.

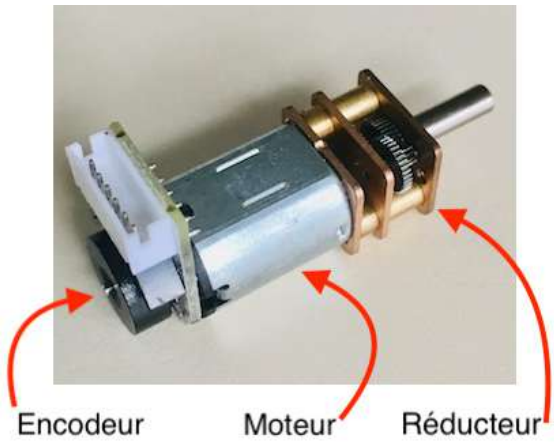


FIGURE 11.1 – Moteur FIT0483.

L'objectif du robot étant de relever des données environnementales en fonction de sa position, les moteurs sont équipés d'encodeurs. Ces encodeurs permettent de connaître le nombre de tours effectués par le moteur. En considérant les dérapages négligeables, cela permet d'envisager de mesurer la position du robot en fonction du mouvement de ses roues (cela s'appelle l'odométrie), bien sûr à condition de connaître la position de départ du robot.

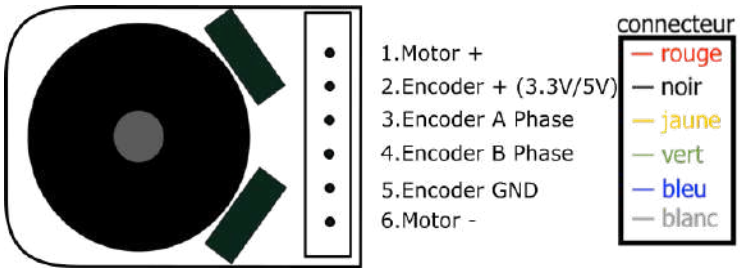


FIGURE 11.2 – Connectique du moteur FIT0483.

La fiche technique du moteur indique [1] :
- Rated Voltage : 6.0 V
- Motor Speed : 15000 RPM
- Gear Reduction Ratio : 100 :1
- Reducer Length : 9.0 mm
- No-Load Speed : 155 rpm@6v
- No-Load Current : 60 mA
- Rated Torque : 0.7 kg.cm
- Rated Speed : 90 rpm@6V
- Current Rating : 170 mA
- Instant Torque : 1.5 kg.cm
- Hall Feedback Resolution : 1400
- Weight : 18g

L'encodeur est constitué de capteurs à effet Hall. Ces capteurs permettent de compter le nombre de tours de l'arbre moteur et donc le nombre de tours effectués par la roue. En fonction du rapport de réduction (100 :1), lorsque l'arbre moteur effectue 100 rotations, la roue tourne d'un tour. Pour un tour de l'arbre moteur, l'encodeur renvoie 14 ticks. Donc pour un tour de roue, l'encodeur renverra 1400 ticks. Le connecteur disponible sur le moteur permet de gérer l'alimentation du moteur ainsi que les signaux issus des encodeurs. le brochage du connecteur est fourni (cf. Figure 11.2). Les broches 2 à 5 sont utilisées pour les encodeurs tandis que les broches 1 et 6 permettront d'alimenter le moteur.

Par rapport à la figure 11.2, les inscriptions sur la carte sont légèrement différentes : le tableau 11.1 donne les correspondances.

11.2 Principe de fonctionnement

Pour piloter le moteur, c'est-à-dire le faire tourner dans un sens ou l'autre, en choisissant la vitesse, nous avons besoin de doser l'apport d'énergie sur les broches Motor+ et Motor-. Le contrôleur de moteur est là pour nous y aider : il sert d'intermédiaire entre la carte Wipy et les moteurs.

Le contrôleur utilisé est le modèle DRV8833 (double pont en H) (Texas Instrument) mais il est fournit sur une carte de marque Adafruit [2]. Ce contrôleur permet de piloter 2 moteurs à

1. voir : `https://www.dfrobot.com/product-1433.html`
2. `https://www.adafruit.com/product/3297`

Numéro (fig. 11.2)	Nom (fig. 11.2)	Code carte	borniers carte de connexion (fig. 10.4)	
			Moteur1-J11	Moteur2-J12
1	Motor+	M2	BOut1	AOut2
2	Encoder +(3.3V/5V)	VCC	3.3V	
3	Encoder A Phase	C2	EncB P18	EncB P13
4	Encoder B Phase	C1	EncA P11	EncA P15
5	Encoder GND	GND	GND	
6	Motor-	M1	BOut2	AOut1

TABLE 11.1 – Câblage du moteur

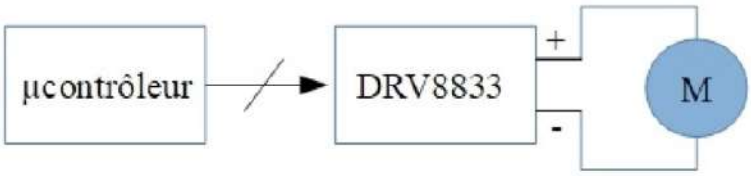


FIGURE 11.3 – Principe de pilotage d'un moteur.

courant continu.

Le principe général de mise en œuvre est décrit par la figure 11.3. Le microcontrôleur génère les signaux nécessaires à la commande du moteur et le contrôleur de moteur DRV8833 gère ensuite le pilotage du moteur.

11.3 Mise en œuvre de connexions

La connexion entre la carte Wipy le contrôleur de moteur DRV883 et les moteurs s'effectue selon le schéma ci-après (cf. Figure 11.4).

Pour faciliter les branchements entre la carte Wipy, le contrôleur de moteurs ainsi que les deux moteurs, le routage des connexions est réalisé par la carte de connexion au travers des deux connecteurs Moteur1 et Moteur2 ainsi que des connecteurs Driver moteur. C'est sur ce dernier connecteur que sera inséré le contrôleur de moteur DRV8833. Le détail du routage des broches du contrôleur de moteur DRV8833 avec celles de la carte Wipy est donné par la Figure 10.4 (page 53).

11.4 Test d'un moteur

Nous allons commencer par utiliser un seul moteur pour prendre en main les commandes de configuration et de mise en mouvement. Il sera alors très facile d'ajouter le deuxième moteur pour que le robot puisse effectuer des mouvements.

La carte contrôleur permet de commander deux moteurs notés `Moteur_Droit` et `Moteur_Gauche`.

Exercice

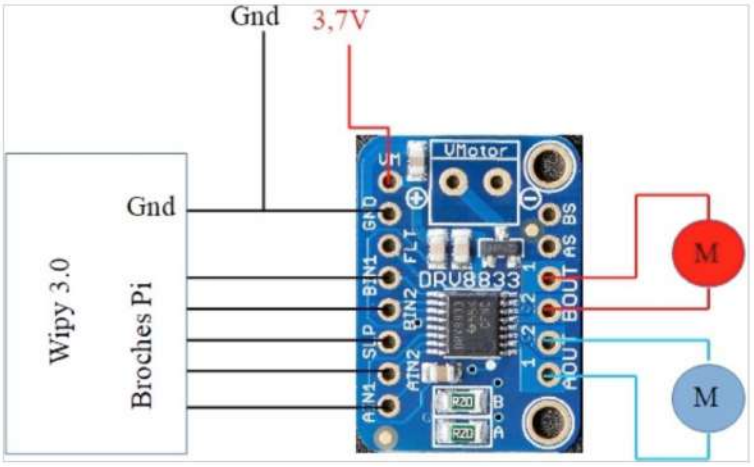


FIGURE 11.4 – Connexion Wipy et Driver DRV8833.

1. Créer un nouveau projet dans Atom : `test_moteur_droit`
2. Ajouter la bibliothèque `DRV8833_V2.py` dans le dossier `lib` du projet que vous venez de créer.
3. Ajouter un fichier `main.py` à la racine de votre projet. Dans ce fichier `main.py`, ajouter la commande d'importation de la bibliothèque :

```
from DRV8833_V2 import *
```

4. Il faut ensuite configurer des variables qui vont contenir les paramètres de connexion d'un moteur : on pourrait utiliser les valeurs directement, par exemple 'P11' mais en cas de déplacement sur une autre broche, il faut reprendre tout le code et remplacer 'P11' par la nouvelle broche. C'est possible mais il y a un risque d'en oublier. Le plus prudent est donc d'utiliser des variables (ou mieux des constantes car le branchement n'est pas sensé évoluer pendant l'exécution du programme). Voici les variables à créer :

```
DRV8833_Sleep_pin = 'P20' # Pin SLEEP
DRV8833_AIN1 = 'P22' # Entree PWM moteur droit : AIN1
DRV8833_AIN2 = 'P21' # Entree PWM moteur droit : AIN2
```

5. L'étape suivante consiste à initialiser le moteur. On créée ainsi une variable Moteur qui sera utilisée pour envoyer des ordres de mouvement. La commande de création a la forme :

```
DRV8833_V2 (In1_pin, In2_pin, sleep_pin, timer_number,
    freq, num_channel_pwm_In1, num_channel_pwm_In2,
    moteur_flag)
```

où les 8 paramètres sont :
— `In1_pin` : entrée PWM 1 DRV8833
— `In2_pin` : entrée PWM 2 DRV8833
— `sleep_pin` : SLP pin pour désactiver les ponts en H du DRV8833

— `timer_number` : dans $\{0, 1, 2, 3\}$. Choix du timer utilisé pour générer le signal pwm. Valeur à utiliser : 1
— `freq` : fréquence du signal pwm : 500 (en Hz : Hertz)
— `num_channel_pwm_In1` : numéro de l'Id du canal PWM associé à la broche `In1_pin`
  — Pour moteur droit : 2
  — Pour moteur gauche : 0
— `num_channel_pwm_In2` : numéro de l'Id du canal PWM associé à la broche `In2_pin`
  — Pour moteur droit : 3
  — Pour moteur gauche : 1
— `moteur_flag` : permettre de désigner le moteur droit ou le moteur gauche à l'aide des constantes définies dans la bibliothèque `DRV8833_V2`
  — `MOTEUR_DROIT_Flag = 1`
  — `MOTEUR_GAUCHE_Flag = 2`

Pour le moteur droit, les entrées PWM sont `AIN1` et `AIN2`. Pour le moteur gauche, il s'agira de `BIN1` et `BIN2`.

Avec les variables créées à la question précédente, la commande d'initialisation d'un moteur devient :

```
Moteur_Droit = DRV8833_V2 (DRV8833_AIN1, DRV8833_AIN2,
    DRV8833_Sleep_pin, 1, 500, 0, 1, MOTEUR_DROIT_Flag)
```

6. Il est alors possible d'utiliser cette variable pour provoquer la rotation du moteur ou bien son arrêt en utilisant la fonction `Cmde_moteur` ou `Arret_moteur` définies au sein de la librairie `DRV8833_V2`

```
Moteur_Droit.Arret_moteur() # Arret du moteur droit
Moteur_Droit.Cmde_moteur(SENS_HORAIRE,
    consigne_rotation_roue)
Moteur_Droit.Cmde_moteur(SENS_ANTI_HORAIRE,
    consigne_rotation_roue)
```

où `SENS_HORAIRE` et `SENS_ANTI_HORAIRE` sont deux constantes, définies au sein de la librairie `DRV8833_V2`, qui permettent de choisir le sens de rotation et `consigne_rotation_roue` est une valeur réelle dans l'intervalle [0.0, 1.78] qui définit la vitesse de rotation en tours par seconde.

À partir de ces commandes, écrire un programme qui répète 10 fois la séquence de mouvements :

```
print('Sequence de mouvements du robot : debut')
consigne_rotation_roue = 0.5
Moteur_Droit.Cmde_moteur(SENS_HORAIRE,
    consigne_rotation_roue)
time.sleep (1)
Moteur_Droit.Cmde_moteur(SENS_ANTI_HORAIRE,
    consigne_rotation_roue)
time.sleep (1)
Moteur_Droit.Arret_moteur()
time.sleep (0.5)
print('Sequence de mouvements du robot : fin')
```

Une fois l'exercice précédent fait et compris, nous pouvons ajouter le deuxième moteur [3] et commencer à préparer le code permettant au robot de bouger.

11.5 Mouvements du robot

Il s'agit de définir les fonctions de déplacement de base du robot, soient :
— `Avancer` ;
— `Reculer` ;
— `Pivoter_droite` ;
— `Pivoter_gauche` ;
Chaque fonction aura comme paramètre la vitesse de rotation des roues. Au sein de chacune des 4 fonctions, il suffit de commander chaque moteur en spécifiant le sens correct de rotation.

Exercice

1. Créer un projet `test_mvt` (inclure la bibliothèque `DRV8833_V2` dans le répertoire `lib`).
2. Créer un fichier `main.py`.
3. en reprenant les éléments vus précédemment, créer 2 variables : `Moteur_Gauche` et `Moteur_Droit` comme précédemment.
4. Écrire alors chacune des fonctions de déplacement selon l'exemple du squelette ci-dessous :

```
def Avancer (consigne_rotation_roue) :
    # Commande du moteur droit
    # Commande du moteur gauche
```

5. Concevoir une séquence de mouvements, en utilisant les fonctions définies plus tôt et vérifier que les mouvements réalisés sont conformes à ceux attendus. Si vous n'avez pas encore fixé les roues sur les axes des moteurs, on peut coller des petits morceaux de papier sur les axes des moteurs pour mieux voir dans quel sens ils tournent !

Les fonctions développées pour l'exercice précédent seront réutilisables dès qu'il s'agira de donner des instructions de déplacement au robot.

11.5.1 Améliorer les déplacements du robot

Il est très probable que la vitesse de rotation de chaque roue ne corresponde pas exactement à la consigne spécifiée. Cela se traduit par le fait que le robot ne décrit pas des trajectoires rectilignes par exemple. Afin de corriger ce problème, nous allons introduire un mécanisme d'asservissement de la vitesse de rotation, similaire à celui d'un régulateur de vitesse au sein d'une voiture.

Pour cela, nous avons besoin de mesurer en temps réel la vitesse de rotation de chacun des moteurs. Cette mesure s'appuie sur des encodeurs placés au niveau de chaque arbre moteur. Ensuite, il faut corriger la consigne de vitesse de rotation de chaque moteur afin d'atteindre la vitesse de rotation souhaitée. On utilise pour cela un correcteur PID (Proportionnel – Intégral

3. Pour connaître les broches, pensez à vous reporter à la figure 10.4 (page 53).

– Dérivé). Le schéma de principe de la commande d'un moteur est alors donné par la figure 11.5.

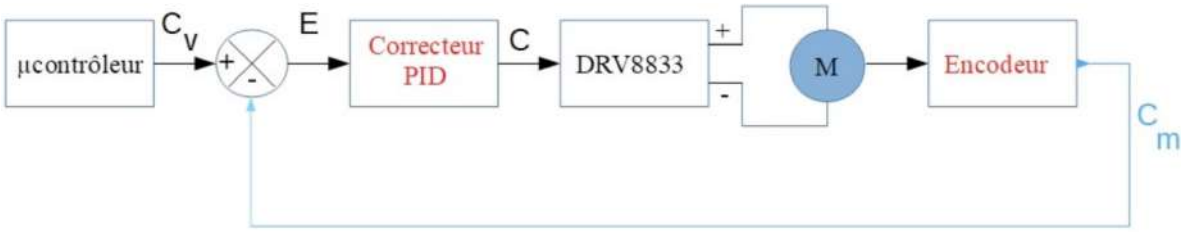


FIGURE 11.5 – Asservissement de vitesse d'un moteur.

où :
— C_v : consigne initiale de vitesse (tours/s)
— C : consigne de vitesse de rotation du moteur (tours/s)
— C_m : Valeur mesurée de la vitesse de rotation du moteur (tours/s)
— E : signal d'erreur, $E(t) = C_v(t) - C_m(t)$ (tours/s)

Exercice Les moteurs utilisés sont équipés d'encodeurs. Ces dispositifs permettent de mesurer la rotation des axes des moteurs. Leur utilisation nécessite l'utilisation de la librairie `ENCODEUR.py`. La correction de la vitesse de rotation de chaque moteur s'appuie sur la librairie `CORRECTEUR_PID.py` en plus de la librairie associée au circuit `DRV8833` de pilotage des moteurs.

1. Créer un nouveau projet dans Atom : `test_moteur_pid`.
2. Ajouter la bibliothèque `DRV8833_V2.py` dans le dossier `lib` du projet que vous venez de créer.
3. Ajouter la bibliothèque `ENCODEUR.py` dans le dossier `lib` du projet.
4. Ajouter la bibliothèque `CORRECTEUR_PID.py` dans le dossier `lib` du projet.
5. Ajouter un fichier `main.py` à la racine de votre projet. Dans ce fichier `main.py`, ajouter la commande d'importation des bibliothèques :

```
from DRV8833_V2 import *
from ENCODEUR import *
from CORRECTEUR_PID import *
```

6. Reprendre les éléments du début de l'exercice de la section 11.5 (page 66.
7. Il faut ensuite configurer les variables qui vont contenir les paramètres de connexion des encodeurs de chaque moteur :

```
# Variables globales pour gestion encodeurs moteurs
Mot_Droit_EncodeurA_pin = 'P15'
Mot_Droit_EncodeurB_pin = 'P13'
Mot_Gauche_EncodeurA_pin = 'P11'
Mot_Gauche_EncodeurB_pin = 'P18'
```

8. Puis il faut définir les paramètres du correcteur PID

```
# Parametres du correcteur PID
Kp = 1.85 # Kp : coefficient proportionnel
Ki = 0.26 # Ki : coefficient integral
Kd = 0.0 # Kd : coefficient derive
Delta_T = 20 # Delta_T : periode d'echantillonnage
    # du correcteur en ms
```

   Les valeurs des coefficients utilisés dépendent des caractéristiques des moteurs utilisés.

9. Reprendre la configuration du circuit DRV8833 effectuée précédemment

10. Créer les variables `Mot_Gauche_Encodeur` et `Mot_Droit_Encodeur`. L'instruction de création de ces variables est de la forme :

```
ENCODEUR (Enc_voieA_pin, Enc_voieB_pin, Pont_H_moteur)
```

    où les trois paramètres désignent :
    — `Enc_voieA_pin` : broche de la carte WiPy qui reçoit les ticks de la voie A de l'encodeur
    — `Enc_voieB_pin` : broche de la carte WiPy qui reçoit les ticks de la voie B de l'encodeur
    — `Pont_H_moteur` : ressources du pont en H du DRV8833 associé au moteur

    L'instruction d'initialisation de l'encodeur du moteur gauche, par exemple, sera :

```
Mot_Gauche_Encodeur = ENCODEUR (Mot_Gauche_EncodeurA_pin,
    Mot_Gauche_EncodeurB_pin, Moteur_Gauche)
```

11. Créer les variables `Moteur_Gauche_Correcteur_PID` et `Moteur_Droit_Correcteur_PID` associées aux correcteur PID de chaque moteur. L'instruction de création de ces variables est de la forme :

```
CORRECTEUR_PID (Kp, Ki, Kd, Delta_T, Encodeur_Mot,
    Moteur_Pont_H)
```

    avec :
    — `Kp` : coefficient proportionnel (voir valeur ci-dessus)
    — `Ki` : coefficient intégral (voir valeur ci-dessus)
    — `Kd` : coefficient dérivé (voir valeur ci-dessus)
    — `Delta_T` : période d'échantillonnage du correcteur en ms (voir valeur ci-dessus)
    — `Encodeur_Mot` : la variable encodeur associée au moteur
    — `Moteur_Pont_H` : la variable moteur associée à l'initialisation du circuit DRV8833

    Par exemple, pour le moteur gauche, nous aurons :

```
Moteur_Gauche_Correcteur_PID = CORRECTEUR_PID (Kp, Ki,
    Kd, Delta_T, Mot_Gauche_Encodeur, Moteur_Gauche)
```

12. Modification des fonctions de déplacements

    Afin de prendre en compte la consigne initiale de vitesse de chaque moteur, il est nécessaire de modifier les fonctions de déplacement décrites précédemment. La modification à apporter sera de la forme :

```
# Routines de deplacement du robot
# consigne_vitesse : tours par seconde
def Avancer (consigne_vitesse) :
    # Commande du moteur droit
    # Commande du moteur gauche
    Moteur_Droit_Correcteur_PID.consigne = consigne_vitesse
    Moteur_Gauche_Correcteur_PID.consigne = consigne_vitesse
```

et pour la fonction Arrêt, nous aurons :

```
def Arret () :
    Moteur_Droit_Correcteur_PID.consigne = 0.0
    Moteur_Gauche_Correcteur_PID.consigne = 0.0
    Moteur_Droit_Pont_H.Arret_moteur ()
    Moteur_Gauche_Pont_H.Arret_moteur ()
```

13. Écrire alors une séquence de mouvements du robot qui pourra prendre la forme suivante :

```
while True :
    Arret()
    time.sleep (0.5)
    Avancer (0.5)
    time.sleep (6)
    Arret()
    time.sleep(0.05)
    Pivoter_Droite (0.3)
    time.sleep(1)
    Arret()
    time.sleep(0.05)
    Pivoter_Gauche (0.3)
    time.sleep(1)
    Arret()
    time.sleep(0.05)
    Reculer (0.6)
    time.sleep(5)
```

11.6 Estimer la position et l'orientation du robot

11.6.1 Principe de l'odométrie

Il s'agit maintenant d'estimer les déplacements effectués par le robot et d'en déduire sa position et son orientation par rapport à sa position de départ. Pour cela, nous allons utiliser les informations issues des encodeurs mis en œuvre dans la partie 11.5.1. L'information issue des encodeurs permettra d'estimer la distance parcourue par chacune des roues du robot, puis d'en déduire les déplacements réalisés par rapport à la dernière position - orientation connue. Ce principe d'estimation s'appelle l'odométrie (cf. Figure 11.6).

À partir de l'estimation des déplacements D_d et D_g associés aux roues droite et gauche, il est possible de calculer la variation de position en Δx et y, ainsi que la variation d'orientation

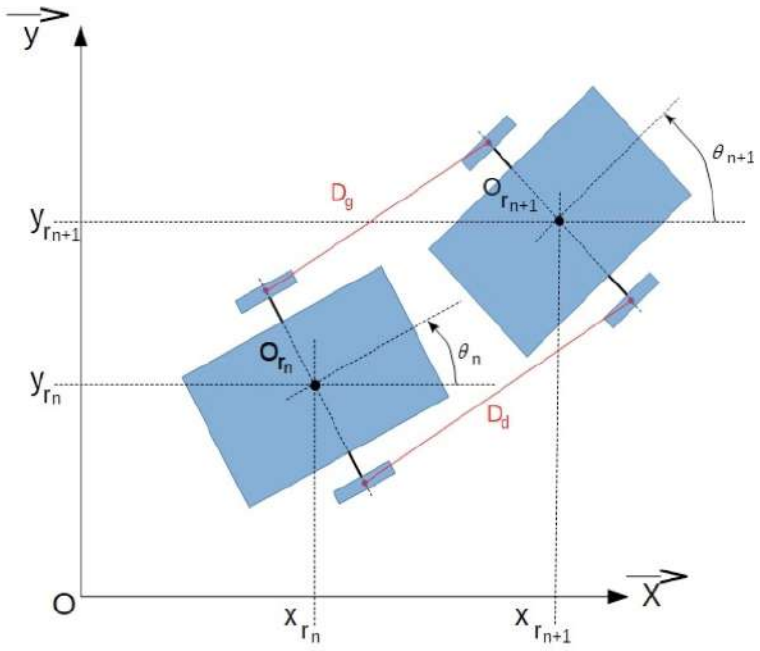


FIGURE 11.6 – Principe de l'odométrie.

$\Delta\Theta$ puis d'en déduire la position et l'orientation absolue par les relations :

$$D_{\text{moy}} = 0,5(D_d + D_g) \quad (11.1)$$
$$\Delta x = D_{\text{moy}} \times \cos\Theta_n \quad (11.2)$$
$$\Delta y = D_{\text{moy}} \times \sin\Theta_n \quad (11.3)$$
$$\Delta\Theta = \Theta_{n+1} - \Theta_n = (D_d - D_g)/L \quad (11.4)$$
$$x_{r_{n+1}} = x_{r_n} + \Delta x \quad (11.5)$$
$$y_{r_{n+1}} = y_{r_n} + \Delta y \quad (11.6)$$
$$\Theta_{n+1} = \Theta_n + \Delta\Theta \quad (11.7)$$

où L désigne la longueur définie par les points de contact des roues sur le sol.

11.6.2 Mise en œuvre

Afin de mettre en œuvre la technique d'odométrie, vous disposez de la librairie `ODOMETRIE.py` qui implémente le calcul de la position-orientation du robot à partir des données issues des encodeurs.

Exercice

1. Créer un nouveau projet dans Atom : `test_odometrie`
2. Au sein du répertoire `lib`, en plus des autres bibliothèques, rajouter la librairie `ODOMETRIE.py`.
3. Ajouter un fichier `main.py` à la racine de votre projet. Dans ce fichier `main.py`, ajouter la commande d'importation des bibliothèques :

```
from ODOMETRIE import *
```

4. Créer la variable `Odometrie` depuis le constructeur de la bibliothèque `ODOMETRIE.py` :

```
Odometrie (x_pos, y_pos, theta, Delta_T,
    Encodeur_Mot_Droit, Encodeur_Mot_Gauche)
```

où :
— `x_pos` : position initiale du robot selon x
— `y_pos` : position initiale du robot selon y
— `theta` : orientation initiale du robot
— `Delta_T` : période d'échantillonnage de l'odomètre en ms
— `Encodeur_Mot_Droit` : ressources associés à l'encodeur du moteur droit
— `Encodeur_Mot_Gauche` : ressources associés à l'encodeur du moteur gauche
Concrètement, cela revient à considérer le code ci-dessous :

```
# Initialisation odometrie
# Position orientation initiale du robot
x_pos = 0.0
y_pos = 0.0
theta = 0.0
print ('Initialisation␣odometre␣:␣begin')
Odometrie = ODOMETRIE (x_pos, y_pos, theta, 15,
    Mot_Droit_Encodeur, Mot_Gauche_Encodeur)
print ('Initialisation␣odometre␣:␣done')
```

La mise à jour de la position et de l'orientation est réalisée toutes les `Delta_T` s (ici 15ms). Lorsque cette durée est écoulée, alors les données de position et d'orientation sont actualisées.

5. Stockage de la position – orientation sur la carte microSD : l'enregistrement des données de position et d'orientation sur la carte microSD est effectué de façon périodique. Pour gérer cette périodicité (ici toutes les 4s), on utilise un drapeau (*flag*) qui autorise cette opération par mise à jour de la variable globale `DATA_Acquisition_FLAG`.

```
# Routine d'interruption pour autoriser
# le stockage des donnees sur la carte microSD
def IT_DATA_SD_Flag (arg) :
    global DATA_Acquisition_FLAG
    DATA_Acquisition_FLAG = True

Timer.Alarm (IT_DATA_SD_Flag, ms = 4000,
    periodic = True)
    # Appel periodique de la fonction
    #IT_DATA_SD_Flag toutes les 4000 ms
```

Finalement, l'enregistrement des données de position et et d'orientation s'effectueront comme suit :

```
Index = 0
while True :
```

```
    # Stockage des donnees sur la carte microSD
    if DATA_Acquisition_FLAG == True :
        Arret()
        time.sleep(0.2)

        # Acquisition date et heure
        # Mesure de distance
         time.sleep(0.002)
        # Mesure de luminosite
        time.sleep(0.002)

        if SD_Flag == True :
            data_registre = repr(Index) + ";"
            for i in range(6) :
                data_registre += repr(date_heure_rtc[i]) + ";"
            for i in range(4) :
                data_registre += repr(Distance[i])+";"
            for i in range(4) :
                data_registre += repr(Luminosite[i])+";"
            data_registre += repr(Odometrie.x_pos)+";"
            data_registre += repr(Odometrie.y_pos)+";"
            data_registre += repr(Odometrie.theta
                    * 180.0 / math.pi)+";"
            data_registre += "\r\n"
            f = open(File_name, 'a')
            f.write(data_registre)
            f.close()
            Index+=1
        DATA_Acquisition_FLAG = False
```

Chapitre 12

Comment détecter les obstacles

Ce chapitre présente la mise en œuvre des capteurs de distance. Ces capteurs permettent au robot d'éviter les obstacles. Ces obstacles pouvant être des murs (obstacles statiques) ou d'autres robots (obstacles dynamiques).

12.1 Introduction

Les capteurs de distance qui ont été retenus sont des VL6180X : c'est un capteur fabriqué par ST
(`https://www.st.com/en/imaging-and-photonics-solutions/vl6180x.html`) et montés sur une carte Pololu (`https://www.pololu.com/product/2489`). Ces capteurs utilisent une technologie *Time-Of-Light* qui consiste à mesurer le temps mis par une émission lumineuse à se refléter sur un obstacle. Ce principe de mesure a l'avantage de ne pas être perturbé par la nature des obstacles : en effet selon la texture/couleur des obstacles, les mesures basées sur la réflectance peuvent varier alors que la durée de l'aller-retour de la lumière n'en dépend pas.

On peut communiquer avec ce capteur en utilisant le protocole I2C qui a déjà été étudié pour le capteur de température. La principale nouveauté vient du fait que nous allons utiliser plusieurs capteurs identiques pour avoir plusieurs informations de distance. Il faut alors configurer chaque capteur pour que les adresses I2C ne rentrent pas en conflit les unes avec les autres.

Nous allons procéder dans le même ordre que pour les moteurs : d'abord nous allons tester la mise en œuvre d'un capteur unique, puis nous « généraliserons » à 3 capteurs.

12.2 Mise en œuvre d'un capteur VL6180X

Par rapport au capteur de température qui utilise également le protocole I2C, il n'y a pas de nouveauté. voici les broches proposées par le capteur :

- `VDD` *: Regulated 2.8 V output. Almost 150 mA is available to power external components. (If you want to bypass the internal regulator, you can instead use this pin as a 2.8 V input with VIN disconnected.)*
- `VIN` *: This is the main 2.7 V to 5.5 V power supply connection. The SCL and SDA level shifters pull the I^2C lines high to this level.*

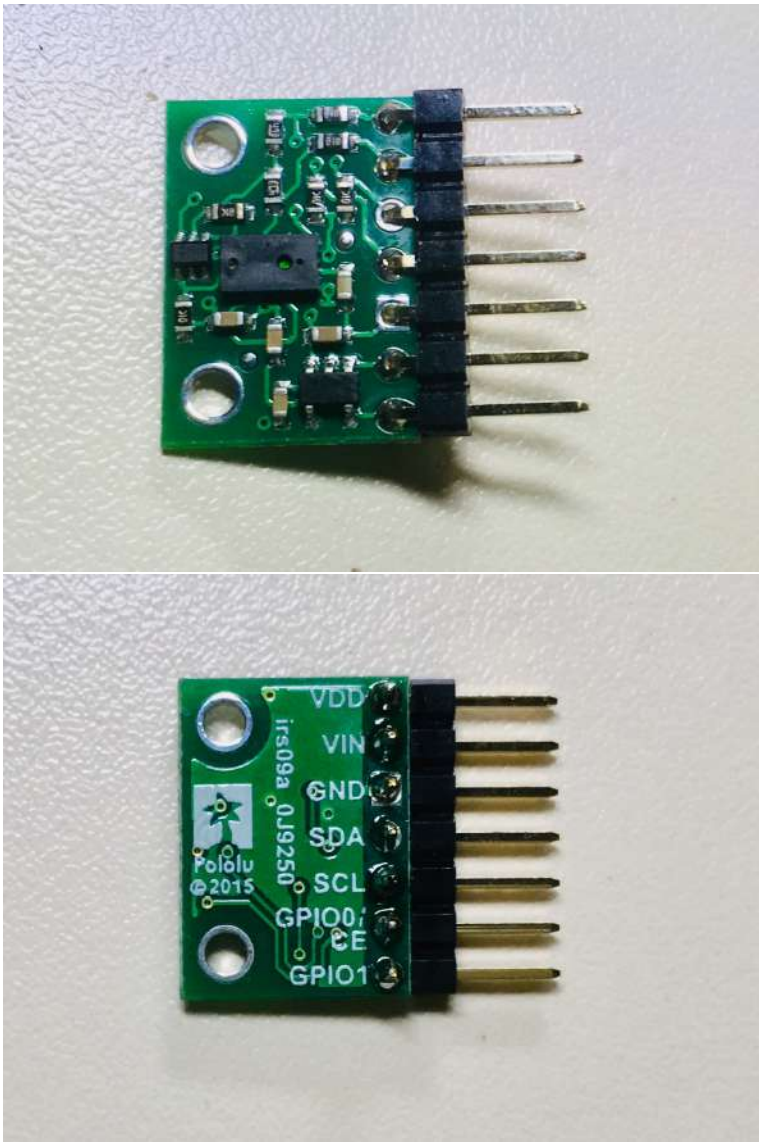


FIGURE 12.1 – Capteur VL6180X (face avant et arrière).

— GND *: The ground (0 V) connection for your power supply. Your I²C control source must also share a common ground with this board.*
— SDA *: Level-shifted I2C data line : HIGH is VIN, LOW is 0 V*
— SCL *: Level-shifted I2C clock line : HIGH is VIN, LOW is 0 V*
— GPIO0/CE *: This pin is configured as a chip enable input on power-up of the VL6180X ; the board pulls it up to VDD to enable the sensor by default. Driving this pin low puts the sensor into hardware standby. After the VL6180X powers up, this pin can be reconfigured as a programmable interrupt output (VDD logic level). This input/output is not level-shifted.*
— GPIO1 *: Programmable interrupt output (VDD logic level). The VL6180X also drives this pin low when it is in hardware standby. This output is not level-shifted.*

Nous aurons besoin des broches VIN et GND (alimentation électrique), SDA et SCL (communication I2C) et GPIO0/CE (activation/desactivation du capteur).

12.2.1 Câblage

Pour faire les tests du capteur, on peut brancher le capteur directement sur la carte de connexion : Sensor1 (voir figure 10.4, page 53). Dans ce cas les broches extérieures (GPIO1 et VDD) ne sont pas utilisées.

Sinon, on peut câbler le capteur en utilisant les broches de la carte d'extension :
— P3 : GPIO/CE
— P9 : SDA
— P10 : SCL
— P24 : VIN
— P25 : GND

12.2.2 Premier test du capteur

Exercice

1. Dans Atom, créer un nouveau projet dans un nouveau dossier test_VL6180X.
2. dans un sous dossier lib de test_VL6180X, recopier la librairie, c'est à dire le fichier VL6180X.py
3. créer le fichier main.py dans le dossier test_VL6180X.
4. vérifier que cela fonctionne en ajoutant la ligne suivante au fichier main.py :

```
from VL6180X import *
```

Cet exercice a permis de mettre en place la suite du programme que nous détaillons ci-dessous.

Il faut déclarer quelques constantes :

```
# Ressource GPIo de la carte WiPy3.0 affectee au controle
# du capteur VL6180X
VL6180X_CE_Pin = 'P3'
# Adressage I2C des capteurs VL6180X : par defaut 0x29 soit 41
VL6180X_I2C_adr_defaut = const(0x29)
```

Ensuite, nous configurons la broche dédiée au contrôle du capteur :

```
VL6180X_GPIO_CE_Pin = Pin(VL6180X_CE_Pin, mode=Pin.OUT)
VL6180X_GPIO_CE_Pin.value(1) # Activer le capteur de distance
```

Nous configurons le bus I2C (en mode maître en utilisant les broches par défaut du Wipy : P9 et P10 et un signal d'horloge à 400kHz) :

```
i2c = I2C(0, I2C.MASTER, baudrate = 400000)
adr = i2c.scan()
print ('Adresse␣peripherique␣I2C␣(1)␣:', adr)
```

Le capteur VL6180X peut maintenant être initialisé :

```
capteur_d_l_VL6180X = VL6180X(VL6180X_I2C_adr_defaut, i2c)
```

Si tout s'est bien passé, non pouvons créer un boucle de lecture des informations du capteur :

```
Index = 0
while True :
    print('Index␣:␣', Index)
    # Acquisition distance et luminosite
    Distance = capteur_d_l_VL6180X.range_mesure ()
    time.sleep(0.002)
    Luminosite = capteur_d_l_VL6180X.ambiant_light_mesure ()
    time.sleep(0.002)
    print ('Distance␣:␣%d' %(Distance))
    print ('Luminosite␣:␣%.1f' %(Luminosite))
    print ('----------------------------------------')
    Index +=1
```

Dans cet exemple, la variable `index` ne sert qu'à vérifier que les lectures se passent bien

Exercice

1. À partir de l'exemple précédent, construire un programme qui affiche la luminosité et la distance détectée par le capteur VL6180X.
2. faîtes varier le 2 grandeurs et vérifiez que le capteur réagit à ces variations.

12.3 Mise en œuvre des 3 capteurs VL6180X

La section précédente s'est concentrée sur l'utilisation d'un seul capteur VL6180X. Pour aider un robot à détecter les obstacles de son environnement, et prendre les bonnes décisions concernant ses déplacements, il peut-être beaucoup plus pratique d'utiliser plusieurs capteurs de distance. Ainsi, le robot sera informé de la position des obstacles en positionnant les capteurs selon plusieurs angles.

Si on souhaite en utiliser simultanément plusieurs capteurs VL6180X, il faut configurer chacun des capteurs avec une adresse différente. On ne peut donc plus se contenter d'utiliser l'adresse par défaut comme dans l'exercice précédent.

Le programme suivant montre la mise en pratique de 3 capteurs de distance VL6180X. Les commentaires expliquent la particularité de la configuration de l'adressage.

```
from VL6180X import *
#-------------------------------------------------------------
# Variables globales pour les 3 capteurs VL6180X
# tableaux de 3 cases initialisees a -1
Distance = [-1, -1, -1]
Luminosite = [-1.0, -1.0, -1.0]
# Nombre de capteurs VL6180X utilises
N_VL6180X = const(3)
# Ressources GPIo de la carte WiPy3.0 affectees au controle
# des capteurs VL6180X
VL6180X_CE_Pin = ('P3', 'P5', 'P6')
# adresse i2c par defaut 0x29 soit 41
VL6180X_I2C_adr_defaut = const(0x29)
# Plage d'adressage I2C des 3 capteurs VL6180X
VL6180X_I2C_Adr = (const(0x2A), const(0x2B), const(0x2C))
#-------------------------------------------------------------
# Initialisation de la broche CE des capteurs VL61800X
# [num capteur] : broche connectee au CE du capteur
# [0] : P6
# [1] : P7
# [2] : P19
print('Config. des broches CE des capteurs VL8160X: debut')
# Liste des variables Pin correspondant aux broches CE
VL6180X_GPIO_CE_Pin = []
for pin in  VL6180X_CE_Pin :
    VL6180X_GPIO_CE_Pin.append(Pin(pin, mode=Pin.OUT))
    # Inhiber chacun des capteurs de distances
    VL6180X_GPIO_CE_Pin[-1].value(0)
print('Config. des broches CE des capteurs VL8160X: fin')
# Initialisation du bus I2C
print ('Configuration bus I2C : begin')
# Init WiPy3.0 en maitre;
# I2C par defaut: P9: SDA et P10: SCL; 400kHz
i2c = I2C(0, I2C.MASTER, baudrate = 400000)
print ('Configuration bus I2C : done')
# Recherche et affichage des peripheriques I2C connectes
adr = i2c.scan()
print ('Adresse peripherique I2C (1) :', adr)
# Init des adresses I2C des capteurs de Distance
# Creation de la liste des objets capteurs de Distance
print('Init. des capteurs de distance-luminosite: debut')
# liste des capteurs de distance : vide a l'initialisation
capteur_VL6180X = []
for i in range (N_VL6180X) :
    # Activer la broche du  capteur VL6180X [i]
    VL6180X_GPIO_CE_Pin[i].value(1)
    time.sleep(0.002) # Attendre 2ms
    # remplir la liste des capteurs de distance
    capteur_VL6180X.append(VL6180X(VL6180X_I2C_adr_defaut, i2c))
    # Init nouvelle adr I2C
    capteur_VL6180X[i].Modif_Adr_I2C(VL6180X_GPIO_CE_Pin[i],
```

```
        VL6180X_I2C_Adr[i], VL6180X_I2C_adr_defaut)
print('Init. des capteurs de distance-luminosite: fin')
adr = i2c.scan()
print ('Adresse peripherique I2C (2) :', adr)

#boucle de lecture des distances+luminosite
Index = 0
while True :
    print('Index : ', Index)
    # Acquisition distance et luminosite
    for i in range (N_VL6180X) :
        Distance[i] = capteur_VL6180X[i].range_mesure ()
        time.sleep(0.002)
        Luminosite[i] = capteur_VL6180X[i].ambiant_light_mesure ()
        time.sleep(0.002)
    print ('Distance : %d %d %d %d' %(Distance[0], Distance[1],
        Distance[2]))
    print ('Luminosite : %.1f %.1f %.1f %.1f' %(Luminosite[0],
        Luminosite[1], Luminosite[2]))
    print ('----------------------------------------')
    Index +=1
```

Ce code montre comment utiliser les listes pour généraliser le stockage des informations. Cela permet de traiter les capteurs identiques en parcourant ces listes.

Exercice

1. Brancher les 3 capteurs VL6180X sur la carte de connexion, en respectant les sens de branchement (voir figure 10.4, page 53).
2. Tester le programme précédent pour vérifier que les 3 capteurs VL6180X fonctionnent en indiquant correctement la distance qui les sépare d'un obstacle.

Ainsi, pour éviter les obstacles, le robot utilise ses capteurs de distance : si une distance devient trop faible (par exemple 2 cm) le robot peut changer de stratégie de déplacement.

Chapitre 13

Intégration

L'intégration consiste à rassembler tout ce que nous avons expérimenté (et validé) dans les chapitres précédents pour que le robot puisse se déplacer et acquérir des données environnementales le long de son trajet.

13.1 Introduction

La programmation du robot doit lui permettre de se déplacer et explorer son environnement tout en récupérant des données environnementales et en évitant les obstacles.

Concrètement il s'agit d'assembler les programmes qui ont déjà été testés pour chaque capteur et les moteurs.

Le schéma (algorithme) général est le suivant :

1. initialiser (configurer) les capteurs, les variables
2. répéter à l'infini :
   (a) lire les informations de température, humidité, etc (chapitre 10)
   (b) enregistrer ces informations avec la position du robot sur la carte SD (chapitre 10)
   (c) pendant 4s faire :
      i. lire les informations de distance(chapitre 12)
      ii. faire avancer le robot en évitant les obstacles pendant 200ms : tourner ou reculer si besoin (chapitre 11)
      iii. évaluer la position du robot (x, y) et son orientation en fonction des mouvements demandés (chapitre 11).

Exercice

1. Construire un programme mettant en œuvre l'algorithme général précédent
2. Vérifier les données inscrites sur la carte SD

Troisième partie

Électronique

Chapitre 14

Introduction

La partie électronique de ce projet se décompose en 4 thématiques qui feront l'objet de 4 séances distinctes :

— L'analyse du fonctionnement d'une résistance et d'une DEL (diode électro-luminescente)
— La gestion de la puissance dans le robot : étude de la batterie
— L'étude de la propulsion
— L'étude et la caractérisation des capteurs

En parallèle de ces quatre études, le montage des éléments (soudure, assemblage) aura lieu pour environ la moitié de chaque séance, afin d'arriver à un robot opérationnel en fin de projet.

Chapitre 15

Fonctionnement d'une résistance et d'une DEL

Afin de visualiser la bon fonctionnement de l'alimentation de notre robot, une résistance et une diode électroluminescente (DEL) vont être placé sur le circuit d'alimentation. Le dimensionnement et le test de ces deux éléments sont indispensables car ils serviront de test visuel pour l'alimentation.

15.1 La résistance

En électricité, il existe 2 notions physiques majoritaires : le courant et la tension.

Le courant représente le débit de porteurs de charges (des électrons dans un métal) au sein d'un conducteur. Si l'on fait une analogie avec un circuit hydraulique, cela représente le débit d'eau au sein d'une canalisation.

La tension représente la différence de potentiels entre deux points d'un circuit (le courant circulant du potentiel le plus haut vers le potentiel le plus bas). Si l'on fait une analogie avec un circuit hydraulique, cela représente la hauteur entre deux points de la canalisation.

La résistance est l'élément électrique permettant de relier ces deux grandeurs de manière proportionnelle suivant la fameuse loi d'Ohm :

$$U = R * I$$

Cet élément électrique est extrêmement utilisé dans tous les domaines de l'électricité. Il est donc primordial d'en connaître le fonctionnement.

15.1.1 Valeurs normalisées

En électronique, la majorité des résistances possède une valeur définie suivant une liste de valeurs bien précise : on parle de résistances normalisées.

La série de valeurs la plus utilisée est la série E24, qui possède 24 valeurs de résistances par décade (c'est à dire par puissance de 10). Le tableau 15.1 donne les différentes valeurs de cette série.

Ces valeurs sont valables pour n'importe quel multiple de 10 : ainsi, on peut avoir des résistances de 1Ω, 10Ω, 100Ω, 1kΩ etc. De même des résistances de 2,2Ω, 22Ω, 220Ω, 2,2kΩ etc.

Série	Valeurs dans la série
E24	1 1,1 1,2 1,3 1,5 1,6 1,8 2 2,2 2,4 2,7 3 3,3 3,6 3,9 4,3 4,7 5,1 5,6 6,2 6,8 7,5 8,2 9,1

TABLE 15.1 – Valeurs normalisées de la série E24

Par contre, pour obtenir une résistance de 2,5kΩ par exemple, il sera nécessaire de combiner plusieurs résistances, car cette valeur n'existe pas dans la série E24.

15.1.2 Code couleur

Sur des résistances standards, il est souvent indiqué la valeur (voire la tolérance et le coefficient de température) sous forme d'anneau de couleur. Chaque anneau représente une caractéristique, comme l'indique la figure 15.1.

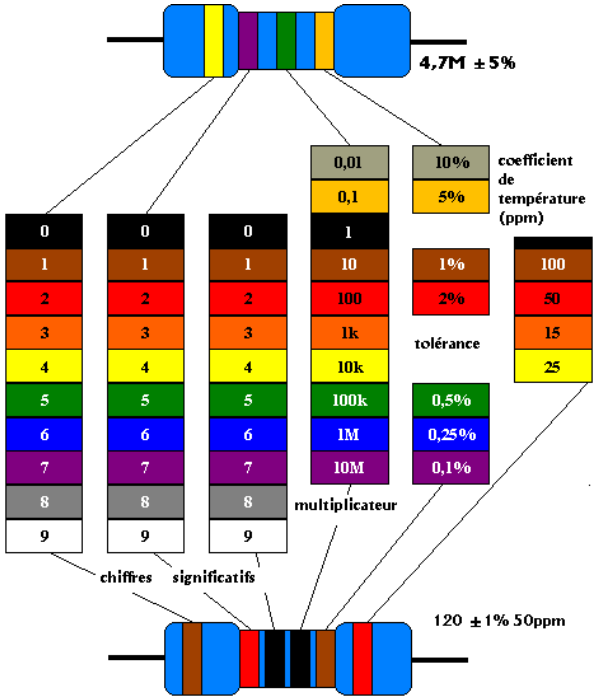


FIGURE 15.1 – Code couleur des résistances

15.2 La diode électroluminescente (DEL)

Une diode est un dipôle électrique (2 pattes) ne laissant passer le courant que dans un sens. Si l'on fait une analogie avec un circuit hydraulique, cela représente une sorte de valve.

Lorsque la diode est passante (courant et tension positifs), le courant évolue de manière exponentiel par rapport à la tension. Il est donc nécessaire d'assurer une tension suffisamment faible à la diode pour éviter d'endommager celle-ci par le passage d'un courant trop important.

Attention ! La diode possède un sens : si vous la montez à l'envers, elle ne sera jamais passante et ne s'allumera donc pas.

Dans le cas d'une diode électroluminescente (DEL, ou LED en anglais), lorsque la diode est passante, elle émet également de la lumière. Le schéma d'une diode électroluminescente dans un circuit électrique, ainsi que l'orientation du courant et de la tension est donné figure 15.2.

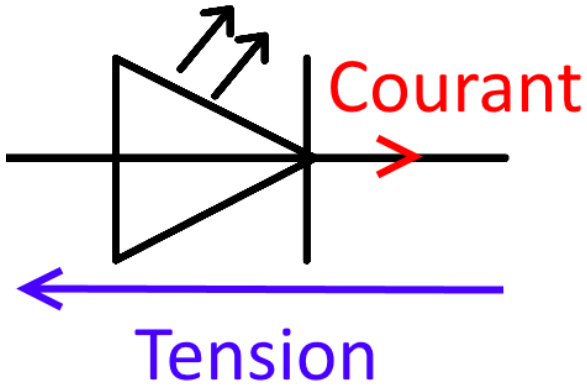


FIGURE 15.2 – Symbole d'une Diode Électroluminescente

La diode utilisée pour le robot est la LED3RL, dont la documentation constructeur est indiquée dans ce lien :

`https://www.sparkfun.com/datasheets/Components/LED/COM-00533-YSL-R531R3D-D2.pdf`

Pour respecter les caractéristiques de tension et de courant de la diode, une résistance sera placé en série avec celle-ci. Pour plus d'explications sur cette méthode, veuillez vous référer aux sites internet suivants :

`https://www.positron-libre.com/cours/electronique/diode/led/alimentation-led.php`

`https://www.astuces-pratiques.fr/electronique/led-et-calcul-de-la-resistance-s`

Chapitre 16

Étude de la batterie

Afin de permettre la mobilité de notre robot sans contact direct vers une source d'énergie fixe (prise électrique), nous utiliserons une batterie.

La batterie utilisée pour notre projet est une batterie Lithium-ions. Elle stocke l'énergie électrique par conversion réversible électrochimique. Les équations des réactions chimiques à l'anode et à la cathode de cette batterie sont les suivantes :

$$\begin{aligned} Li_{1-x}CoO_2 + xLi^+ + xe^- &\underset{\text{charge}}{\overset{\text{décharge}}{\leftrightharpoons}} LiCoO_2 \\ Li_xC_6 &\underset{\text{charge}}{\overset{\text{décharge}}{\leftrightharpoons}} C_6 + xLi^+ + xe^- \end{aligned} \tag{16.1}$$

Le symbole x indique que la cathode et l'anode sont constituées d'un matériaux cristalin dans lequel s'insère un nombre d'ions lithium (on parle de réaction d'intercalation) : leur nombre dépend de l'état de charge de la batterie (idéalement, à batterie totalement pleine, x=1, à batterie complètement déchargée, x=0).

16.0.1 Tension de batterie

Une batterie de type Lithium-ion possède une tension moyenne à vide (c'est à dire non branchée) de 3,7V. Celle-ci varie légèrement en fonction de son niveau de charge : supérieure à 4V environ lorsqu'elle est entièrement chargée, et pouvant descendre à moins de 3V lorsqu'elle est totalement déchargée. La figure 16.1 montre la variation de la tension de la batterie en fonction de son état de charge. Cette tension dépend également de la température, de l'hygrométrie et de la pression ambiante.

Lorsqu'elle est en charge (c'est à dire qu'elle envoie du courant dans un circuit électrique), la tension de la batterie va varier en fonction du courant qu'elle produit. Cette variation, bien que normalement non linéaire, peut être modélisée en simple approximation par une équation linéaire :

$$U_{batterie} = U_{\text{batterie à vide}} - r \times I_{batterie} \tag{16.2}$$

Ainsi, une batterie peut être modélisée par une source de tension (la tension de la batterie à vide), à laquelle ont soustrait une tension de charge, proportionelle au courant et à une résistance de charge r. La figure 16.2 donne le schéma électrique du modèle de la batterie.

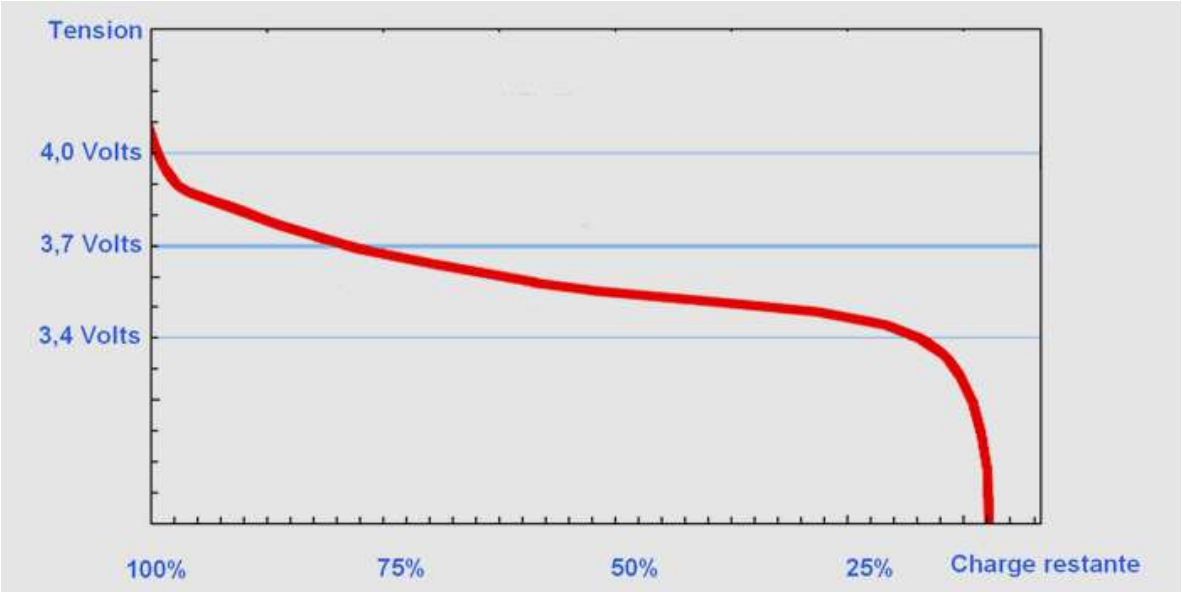


FIGURE 16.1 – Courbe Tension à vide de batterie vs état de charge

16.0.2 Capacité de la batterie

Il existe 2 caractéristiques importantes concernant le choix d'une batterie.

La première concerne la puissance qu'est capable de fournir celle-ci. En électricité, la puissance se traduit pas le produit de la tension par le courant :

$$P = U \times I \quad \text{en Watt (W)} \tag{16.3}$$

En réalité, cette puissance maximum est liée à la vitesse maximale de la réaction chimique de la batterie. Elle n'est donc pas lié à la tension, mais uniquement au courant. C'est pourquoi on parle souvent de courant maximum et non de puissance maximum. Attention ! Cette limite de courant peut-être différent en charge et en décharge.

La deuxième caractéristique est la capacité de la batterie, c'est-à-dire l'énergie maximum qu'elle est capable de stocker. L'énergie électrique correspond à la quantité de puissance débitée par la batterie durant un temps donnée. Elle se traduit mathématiquement par l'équation suivante :

$$E = \int_{(T)} Pdt = \int_{(T)} U \times Idt \tag{16.4}$$

L'unité standard de l'énergie est le Joule (J). Cependant, le Joule représentant la consommation d'1 Watt pour 1 seconde, c'est une unité peu adapté à des systèmes actuels (la moindre installation consommant des MégaJoules, voire des GigaJoules). Ainsi, les documentations et informations ne ciblant pas toujours des experts, l'unité la plus souvent utilisé est le Watt-heure (consommation d'1 Watt pendant 1 heure). Le Watt-heure vaut donc 3600 Joules. De plus, pour la même raison que pour la puissance précédemment, la quantité d'énergie stockée par la batterie dépend du nombre d'électrons qu'elle est capable de stocker, et donc ne dépend

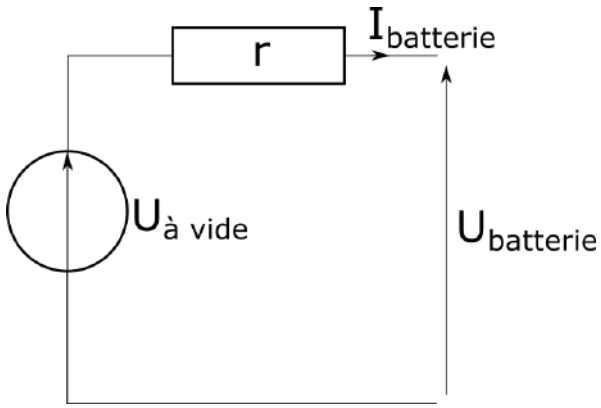


FIGURE 16.2 – Symbole d'une Diode Électroluminescente

pas de la tension, mais uniquement du courant. C'est pourquoi le milieu industriel a introduit la notion de capacité de la batterie, dont les unités sont des Ampères-heure. Cela représente la quantité de courant qu'est capable de fournir la batterie pendant 1h (pour une décharge complète) : pour un courant moitié moindre, elle durerait 2h, etc. Il s'agit donc bien d'une représentation de l'énergie contenue dans la batterie, mais sous une unité différente. Ainsi, pour repasser aux unités normalisées, on n'oubliera pas de multiplier cette capacité par la tension moyenne de la batterie pour passer en Watt-heure, puis par 3600 pour passer en Joules.

La capacité C de notre batterie est de 2000 mAh. Elle est capable de fournir un courant max de l'ordre de 1,5*C en décharge, et absorbe au maximum 0,5*C en charge.

Chapitre 17

Etude de la propulsion

La propulsion de notre robot se fait à travers deux servo-moteurs électriques. Ceux-ci se pilote chacun à travers un hacheur réversible de type "pont en H", inclus sur la carte driver DRV8833.

Sur chaque moteur, un tachymètre a été intégré à l'axe afin d'avoir un retour sur la position de l'arbre moteur.

Chapitre 18

Caractérisation des capteurs

Le robot est constitué de différents capteurs, permettant d'analyser sa position dans l'espace, les obstacles devant lui, ainsi que son environnement.

Trois capteurs de distance VL6180X, sont utilisés pour détecter et éviter les obstacles.

Le capteur de pression, température et humidité est le BME280, dont la documentation constructeur peut se retrouver ici :

`https://www.embeddedadventures.com/datasheets/BME280.pdf`

Chapitre 19

Principe de la soudure en électronique

La soudure de composant électrique (on parle normalemnet de brasure, car on ne fait pas fondre les pates du composant, mais un matériau différent, ajouté en plus) est une technique indispensable pour la réalisation de cartes électronique. Bien que réalisé industriellement depuis de nombreuses années, il est important pour un électronicien de savoir souder, pour faire notamment des prototypes, des réparations ou des améliorations sur des cartes existantes.

Au sein du projet, de nombreux composants, borniers et fils sont à souder pour réaliser le robot. Nous verrons ici uniquement la soudure de composants traversants, la soudure de composant en surface étant beaucoup plus fastidieuse.

19.1 Matériel nécessaire

Pour réaliser une soudure électrique, il vous faut :

— Un fer à souder : il en existe de différentes formes, à air chaud, réglable en température, à panne interchangeable, pistolet à souder, sans fil...
— De la brasure ou fil à souder : à base d'étain et parfois de plomb (interdit dans l'industrie), c'est l'élément qui va fondre et faire adhérer le composant au circuit. Il joue également le rôle de conducteur à l'interface.
— Une pince à couper les surplus de composants soudés

D'autres éléments non indispensables mais très pratiques sont souvent utilisé :

— Une tresse à dessouder : c'est une tresse métallique qui absorbe la brasure fondue. Elle permet de dessouder lors d'une soudure ratée.
— Une pompe à dessouder : même rôle que la tresse, la pompe à dessouder aspire la brasure fondue.
— Une troisième main : outil constitué de pinces permettant de tenir les éléments à souder en place, tel une "troisième main", d'où le nom.
— Un nettoyeur à sec : sorte de grosse bobine de métal doré servant à nettoyer son fer à souder
— Une lampe et une loupe, pour mieux voir l'opération
— Un extracteur de vapeur : pour évacuer les gaz émis lors de la soudure, qui sont néfastes à la santé

19.1.1 Le fer à souder

Le choix du fer à souder est important pour réaliser une bonne soudure. Celui-ci dépend du type de soudure que l'on souhaite effectuer. Pour une soudure de composants traversants un fer à souder classique est préférable. Dans le cas d'une soudure de composant de surface, un fer soufflant à air chaud peut être plus pratique.

FIGURE 19.1 – Fer à air chaud

Le prix d'un fer à souder peut être extrêmement variable (d'une dizaine d'euros à plusieurs centaines). Il dépend des caractéristiques de celui-ci :

— Puissance : 15 à 20W pour les moins cher, qui ne permettent que des soudures très ponctuelles (montée en température lente).
— Régulation de la température : pour réaliser une bonne soudure, il est préférable d'avoir un fer à température régulée, afin que le matériau soit chauffé toujours à la même température.
— Panne interchangeable : la panne est l'embout du fer. A force de soudure, celui-ci s'oxyde et devient moins efficace. Il peut être intéressant de pouvoir le changer sans changer le fer en entier. De plus, la taille de la panne dépend du la taille de la soudure que l'on souhaite faire.

19.1.2 La brasure ou fil à souder

Les matériaux les plus utilisés pour souder sont la pâte à braser (une pâte semi-liquide utilisée pour les composant à souder en surface), et le fil à souder. Pour ce dernier, on peut prendre en compte plusieurs paramètre dans le choix de celui-ci :

— L'épaisseur : comme pour la taille de la panne, celui-ci dépend de la taille de la soudure que vous souhaitez faire. Un fil plus fin sera plus pratique pour les petites soudures,

FIGURE 19.2 – Éléments d'un fer à souder

mais s'utilisera beaucoup plus vite pour les grosses soudures. Pour les soudures les plus classiques, du fil 0,7mm ou 1mm de diamètre sont les plus simples à utiliser.

— Le flux de soudure : la plupart des fils à souder vendus pour faire de l'électronique sont creux. A l'intérieur, une sorte de cire appelée *flux de soudure* est incorporée. Elle sert d'antioxydant et aide à la bonne prise de votre soudure. Il est donc très recommandé de prendre du fil à souder avec flux de soudure intégré.

— Teneur en plomb : pour des raisons écologiques, les fils à souder contenant de plomb sont interdits pour la production industrielle. Ils sont alors remplacé par une proportion d'étain plus importante. Ils sont cependant toujours toléré dans le cadre amateur ou pédagogique, car ils sont plus facile à souder (température de fusion plus basse). Il faut tout de même faire attention au dégagement gazeux plus toxique avec du fil au plomb (plate-forme de soudage aérée ou hotte aspirante).

19.1.3 Nettoyeur de panne

Il existe deux type de nettoyeur de panne : une éponge mouillée et un "nettoyeur sec", sorte d'éponge à gratter métallique. Cette dernière, bien qu'un peu plus chère est préférable car elle nettoie mieux a panne, et ne modifie pas la température du fer lors du nettoyage.

Il est indispensable de nettoyer régulièrement son fer (idéalement, à chaque opération où du fil à souder vient toucher la panne) car si une grosse pellicule de soudure se retrouve sur la panne, le fer ne va plus chauffer correctement. De plus l'oxydation de la panne sera plus importante et devra être changé prématurément.

19.1.4 Éléments pour dessouder

Il existe deux objets principaux pour dessouder : la pompe à dessouder et la tresse à dessouder.

La pompe à dessouder vient aspirer la soudure fondue par le fer à travers un bec. Bien que facile à utiliser, elle laisse parfois une partie de la soudure sur le circuit.

La tresse à dessouder est un fil métallique tressé que l'on applique sur la soudure, et que l'on chauffe avec le fer. Lorsque la température est suffisamment élevée pour faire fondre la soudure, celle-ci est aspirée par capillarité sur la tresse. Cette méthode est un peu plus longue

FIGURE 19.3 – Nettoyeur à sec

mais souvent plus efficace. Elle permet également de dessouder plusieurs pattes en même temps.

FIGURE 19.4 – Pompe à dessouder et Tresse à dessouder

19.2 Sécurité

Avant de commencer à souder, il est nécessaire de respecter un minimum de précaution pour garantir votre sécurité, car il s'agit d'une opération relativement dangereuse. La manipulation d'éléments chaud, en fusion ainsi que les vapeurs peuvent entraîner des conséquences graves.

C'est pourquoi il vous est demandé de **ne jamais manipuler seul**. La présence d'un enseignant dans la pièce et d'un camarade à vos côté est obligatoire lors de la réalisation de soudure dans ce projet. Tout manquement aux règles de sécurité entraînera des sanctions. La figure 19.5 vous indique les consignes de sécurité minimum à avoir lors d'une opération de soudure électronique.

Les vapeurs de soudure sont toxiques, il est donc indispensable de travailler dans un espace suffisamment aéré, et de ne **pas respirer les fumées** lors de la soudure. Il est aussi important de se **laver les mains** à la fin de vos travaux, car de l'étain ou du plomb (matériaux toxiques à l'ingestion) peuvent s'être glissé sous vos ongles ou sur votre peau.

Le fer à souder est extrêmement chaud (de l'ordre de 300 ° C, soit presque deux fois plus que de l'huile de friture bouillante). Sa manipulation doit donc être faite avec une extrême prudence : évitez la présence de personnes trop proches que pourraient gêner la manipulation du fer ou être brûlé par inadvertance. Manipulez toujours avec des vêtements à manches courtes ou plaquées et cheveux attachés. Ne poser jamais le fer ailleurs que sur son support, et **ne laisser jamais le fer allumé sans surveillance**. Il pourrait créer un incendie.

Le flux de soudure chaud peut parfois sauter (comme une huile bouillante) et potentiellement vous brûler. Si cet incident reste mineur sur la peau, elle peut s'avérer plus grave dans les yeux : **il est donc indispensable de mettre des lunettes de sécurité ou d'avoir un écran de protection** lors de l'utilisation du fer à souder.

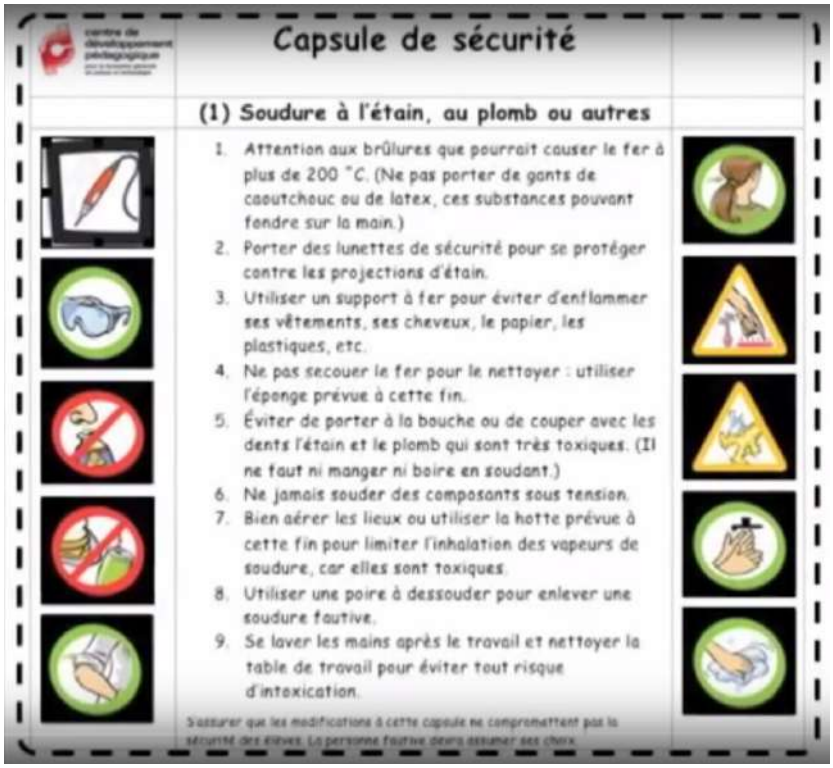


FIGURE 19.5 – Consignes de sécurité

Même s'ils ne sont pas indispensables et peuvent gêner à la manipulation, l'utilisation de gant à protection thermique peut être une sécurité supplémentaire pour éviter les brûlures aux mains. Attentions ! Les gants en plastique ou non adaptés aux températures de soudure (450 ° C max) sont absolument à proscrire. Ils pourraient s'enflammer et énormément aggraver la situation.

Enfin, il est important de laisser son espace de travail propre et bien rangé. Il est interdit

de boire ou de manger lors des opérations de soudure, et il est important de ne rien laisser traîner durant la soudure. En fin de séance, l'ensemble de matériel doit être nettoyé et ranger à la bonne place.

19.3 Effectuer une soudure

La première étape de la soudure consiste à placer correctement le composant à souder sur la carte électronique. Faites attention au sens du composant : la soudure est généralement face verso, et le composant face recto. Celui-ci doit être placé au plus près de la carte. Faites attention aux composants possédant plusieurs pattes : il est important de positionner l'ensemble des pattes correctement avant soudure. Une fois une patte soudée il est souvent difficile de replacer le composant, et il ne reste plus qu'à dessouder.

N'hésitez pas à utiliser une troisième main pour fixer votre composant de manière stable sur la carte. Lors de la soudure, **ne tenez pas le composant avec vos doigts** : celui-ci va chauffer, et vous risquez de vous brûler.

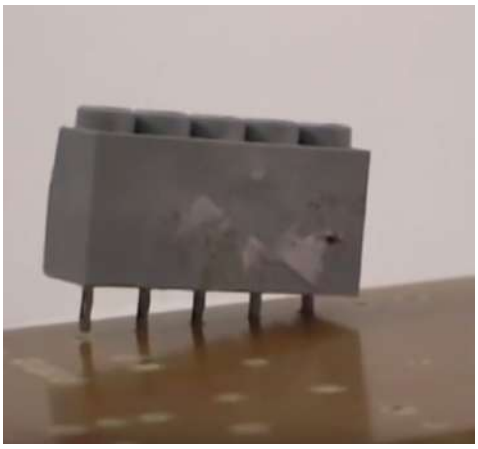


FIGURE 19.6 – Composant mal placé sur un carte

Une fois le composant correctement placé, vous aller pouvoir souder. Un fer à souder doit être aux alentours de 300 °C pour souder correctement de l'étain. Si votre fer est réglable, mettez le à cette température.

Pour vérifier la température du fer faites fondre une faible quantité d'étain sur la pointe de celui-ci : si elle fond instantanément, c'est que le fer est suffisamment chaud. Cette opération a également pour but de légèrement étamer la pointe du fer pour une meilleure surface de contact lors de la soudure. Attention ! Il ne faut pas mettre trop de soudure sur la pointe. Si vous constater un "paquet" de soudure sur la pointe de votre fer, nettoyez le avant de souder.

Placez ensuite le fer chaud à l'interstice entre la carte et la patte du composant. Il est important de bien être en contact avec ces deux éléments, car c'est la chaleur qui va conduire la soudure à se fixer sur ceux-ci. Maintenez le fer pendant 3 à 5 secondes avant d'avancer le fil de soudure.

Mettez le fil de soudure en contact avec la pointe du fer, au plus près du contact entre le fer, la carte et le fil du composant. Laissez fondre une petite quantité d'étain et retirer le fil de

soudure. Sans retirer le fer, visualisez votre soudure : si la quantité de soudure est suffisante, alors l'étain devrait enrober l'ensemble de la pastille de la carte électronique. Sinon, rajouter un peu d'étain en replaçant votre fil de soudure. Une fois la quantité satisfaisante, retirez le fer. Une bonne soudure doit avoir la forme d'un dôme dont le sommet est la patte du composant.

La figure 19.7 illustre graphiquement les différentes étapes d'une soudure.

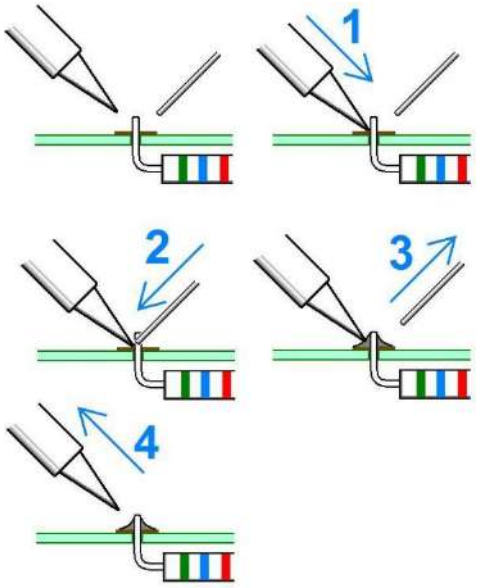


FIGURE 19.7 – Étapes d'une soudure

19.3.1 Mauvaises soudures

La figure 19.8 illustre des mauvaises soudures. Il est nécessaire de refaire ces soudures car elles pourraient entraîner un mauvais fonctionnement du circuit final. Ces mauvaises soudures peuvent être dues à plusieurs facteurs : une mauvaise connexion de la soudure à la carte, de la soudure à la patte du composant, ou encore une soudure trop large qui déborde sur une autre et provoquerait un court-circuit.

Dans l'ensemble de ces cas, il est nécessaire de refaire proprement la soudure, et généralement de dessouder l'ancienne.

19.4 Dessouder

En cas d'erreur après une soudure (mauvais placement du composant, soudure mal faite...), il faut dessouder la ou les pattes du composant. Pour se faire, on peut utiliser soit une pompe à dessouder, soit une tresse.

19.4.1 Avec une pompe à dessouder

Armez votre pompe à dessouder en appuyant sur la tige. Profitez-en pour vérifier que votre pompe est bien vide.

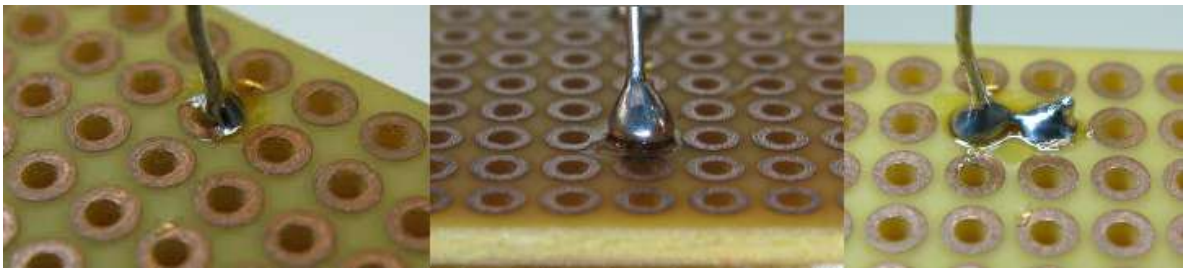


FIGURE 19.8 – Mauvaises soudures

Amenez votre pompe au niveau de la soudure et positionner votre fer chaud sur la soudure pour la faire fondre. La pompe et le fer doivent former un "V" au dessus de la carte. Faites attention à ne pas toucher la pompe avec le fer pour ne pas endommager l'embout de celle-ci. La figure 19.9 vous indique le placement idéal de vos outils.

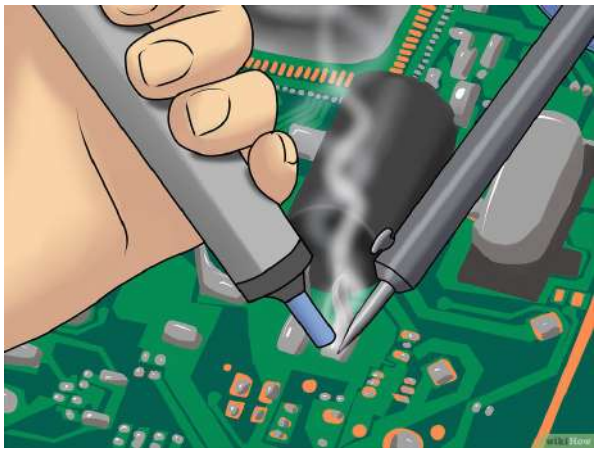


FIGURE 19.9 – Placement de la pompe pour dessouder

Une fois la soudure fondue, appuyez sur le bouton de la pompe pour enclencher l'aspiration de la soudure.

Il se peut que vous ayez besoin de réitérer l'opération plusieurs fois pour obtenir une carte propre. Ne laisser jamais des bouts de soudure sur la carte (même si vous pouvez enlever le composant) car ceux-ci viendront polluer une nouvelle soudure à cet endroit.

19.4.2 Avec une tresse à dessouder

Dévider une partie de la tresse à dessouder. Si l'extrémité de la tresse vous semble avoir déjà été utilisé (couleur différente), couper la tresse pour utiliser un bout vierge.

Poser l'extrémité de la tresse sur la patte à dessouder, puis poser le fer chaud sur la tresse (voir figure). Attendez une dizaine de secondes : vous devriez voir la tresse absorber la soudure. N'hésitez pas a avancer la tresse car celle-ci se rempli vite. Par contre, faites attention à ne pas toucher une autre soudure dans l'opération pour ne pas endommager celle-ci.

Au bout d'une dizaine de secondes (pas plus, pour ne pas endommager le composant que l'on cherche à dessouder), retirez le fer et la tresse. Si la soudure n'a pas été complètement absorbée, réitérez l'opération.

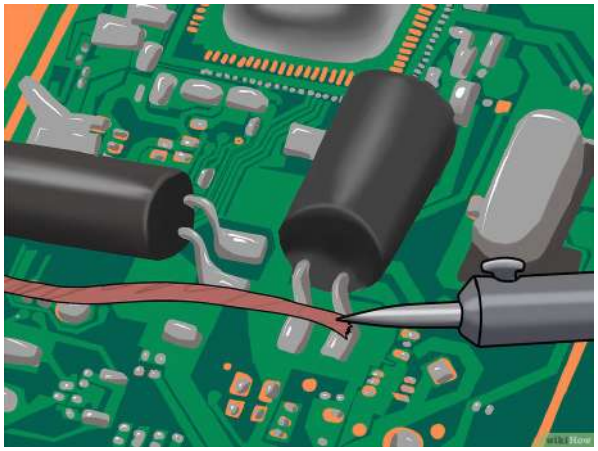


FIGURE 19.10 – Dessouder à la tresse

19.5 bibliographie

Il existe de nombreux ouvrages expliquant la méthodologie de soudure et dessoudure en électronique (comme *l'électronique pour les débutants* de Rémy Mallard - édition Elektor, ou encore *Apprendre L'électronique Fer À Souder En Main* de Jean-Pierre Oehmichen - édition Dunod).

Cependant il est également possible de se procurer ses informations sur internet facilement. Les explications précédentes s'appuient sur les sites internet suivants :

- *Comment souder à l'étain* - Diego Yourself : `http://www.diegoyourself.com/index.php?article=electronique_souder&page=0`
- *Conseils soudure à l'étain* - Interface-Z : `https://www.interface-z.fr/conseils/soudure.htm`
- *Comment Dessouder* - wikiHow : `https://fr.wikihow.com/dessouder`

Plusieurs vidéos sont également intéressantes sur le sujet :

- *Apprendre à souder en 5 minutes* - U=RI : `https://www.youtube.com/watch?v=sXKvqmXQI8o`
- *La soudure à l'étain* - CDP ScienceTechno : `https://www.youtube.com/watch?v=cQipqFqEpWA`

Quatrième partie

Aménagement-Environnement

Chapitre 20

Introduction

Les robot a parcouru un territoire et enregistré différentes informations (température, luminosité) et identifié des obstacles. Cette démarche représente, de manière simplifié bien sûr, ce qui peut se passer dans la réalité où cet exercice se réalise à plus grand échelle, en 3D (chaque point est caractérisé en x,y et z), au moyen de satellites et de véhicules de type "google street view". C'est là la version moderne et numérique d'un processus de connaissance d'un territoire qui a une longue histoire, notamment incarnée en France par les travaux de l'abbé Picard, relayé par 4 générations de membres de la famille Cassini au début du 17ième siècle. Ce processus de connaissance et de représentation de l'espace est la cartographie [1].

On comprendra que tout cela est un préalable à l'activité des aménageurs et des urbanistes. A notre époque du tout numérique et du "big data", les "Systèmes d'Information Géographiques" (SIG)" ce sont substituées aux cartes et permettent une connaissance et des analyses thématiques très fines. La connaissance de tout phénomène ayant une dimension spatiale s'en trouve à la fois facilitée, mais aussi complexifiée au regard de la quantité de données qui alimentent ces systèmes.

Dans le volet "aménagement" de ce projet multidisciplinaire, nous nous plaçons en post-traitement des activités de conception, de réalisation, de gestion des déplacements du robot et de collecte des données recueillies au moyen de ses capteurs. Par différentes étapes de qualification, ces données vont alimenter un Système d'Information Géographique (SIG). A partir de cet outil, nous allons penser l'organisation du territoire au moyen d'un des outils les plus opérationnels de l'urbanisme, le Plan Local d'Urbanisme (PLU). Il s'agit de définir des "zones" qui présentent une certaine homogénéité et auxquelles à attribue des usages en rapport. On caractérise ainsi notamment des zones "urbaines", des zones "agricoles" ou des zones "naturelles". Dans chacune, des règles spécifiques vont conditionner les possibilités de nouvelles implantations ou de nouvelles constructions. On met ainsi en place un outil qui va régir l'évolution raisonnée du cadre de vie des habitants d'un territoire.

Les objectifs de cette partie sont :

— Objectif 1
— Objectif 2
— Objectif 3
— Objectif 4

1. Pour une approche accessible de cette histoire voir Denis Guedj - Le mètre du monde ou La méridienne - en poche.

La répartition des 4 séances est la suivante :

1. Séance 1
2. Séance 2
3. Séance 3
4. Séance 4

Chapitre 21

Importation des données collectées par le robot dans un SIG

Le robot collecte des données en provenance de différents capteurs. Il est en mesure d'estimer sa localisation dans l'espace par odométrie. Cette méthode de positionnement relatif par rapport à un point de départ connu (x_0, y_0) se base sur le comptage de la distance (nombre de tours de roues) effectué par le robot. Ainsi le robot est capable d'approximer sa localisation dans l'espace (x, y). Cette localisation est complémentée par une information temporelle pour former une position (x, y, t) et des données collectées par différents capteurs (distance et angle vers un obstacle, chaleur, luminosité) à cette position (x, y, t, d, a, c, l). Chaque position est ensuite sauvegardée par le robot dans un fichier CSV stocké sur une carte mémoire (MicroSD). L'objectif de cette section consiste à traiter les données collectées par le robot pour les afficher dans le SIG.

Cinquième partie

Annexes

Annexe A

Liste du matériel

Nom	Modèle	Quantité	image
Batterie	3.7V, 2000mAh	1	

Annexe B

Mise à jour du *firmware* pour le Wipy

Il est nécessaire de télécharger l'utilitaire de mise à jour fourni par Pycom, selon le type de système d'exploitation dont on dispose. La procédure est illustrée pour Windows. Lorsque l'utilitaire est téléchargé, il faut appliquer la procédure suivante :

1. Déconnecter la carte du port USB ;
2. Configurer la carte en mode mise à jour : relier la broche G23 à la broche GND ;
3. Reconnecter la carte sur un port USB. Le matériel est alors en mode « Mise à jour » ;
4. Lancer l'utilitaire de mise à jour.

Les écrans successifs qui apparaissent sont donnés par les figure B.1 à B.10.
LES IMAGES ONT ÉTÉ DÉSACTIVÉES TEMPORAIREMENT

FIGURE B.1 – Mise à jour du firmware de la carte Wipy2.0

FIGURE B.2 – Mise à jour du firmware de la carte Wipy2.0

FIGURE B.3 – Mise à jour du firmware de la carte Wipy2.0

FIGURE B.4 – Mise à jour du firmware de la carte Wipy2.0

FIGURE B.5 – Mise à jour du firmware de la carte Wipy2.0

FIGURE B.6 – Mise à jour du firmware de la carte Wipy2.0

FIGURE B.7 – Mise à jour du firmware de la carte Wipy2.0

FIGURE B.8 – Mise à jour du firmware de la carte Wipy2.0

FIGURE B.9 – Mise à jour du firmware de la carte Wipy2.0

FIGURE B.10 – Mise à jour du firmware de la carte Wipy2.0

Annexe C

Ressources bibliographiques

Les sites web suivants vous permettrons de compléter les informations données dans ce documents :

— **Pycom Documentation** : `https://docs.pycom.io/`
ce site contient toutes les informations sur la programmation des module Wipy.