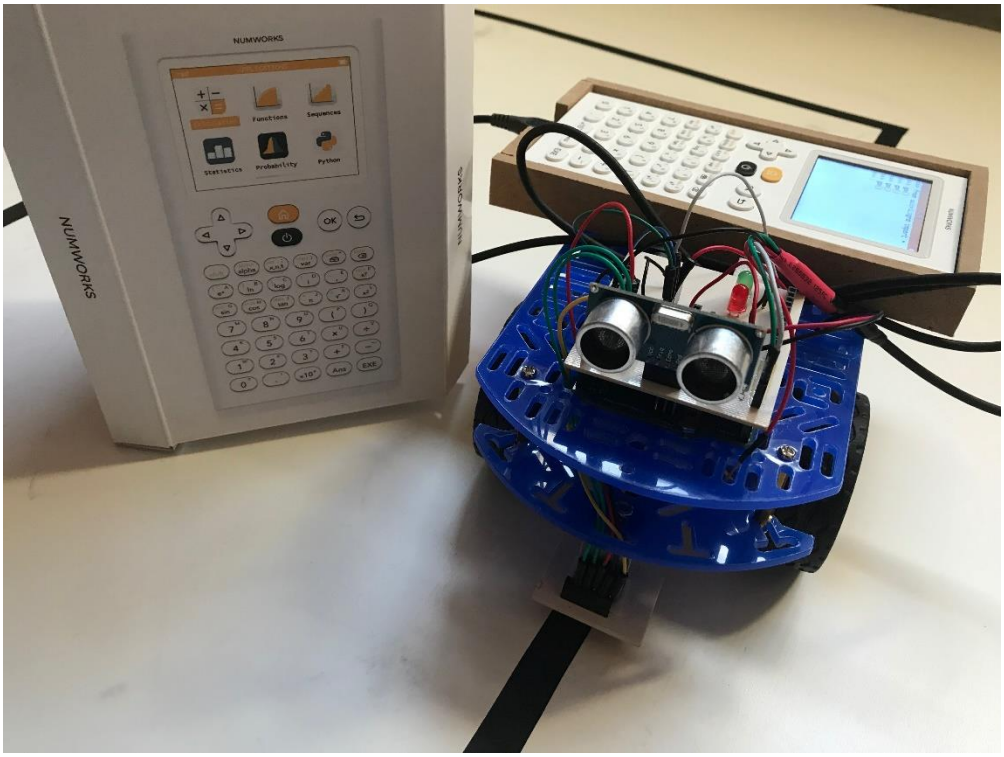
Numworks & Robot



Projet réalisé par Maxime Créteur

Projet P2 IMA4 2018/2019

Tuteurs : X. Redon / A.Boé / T. Vantroys

Polytech Lille département Informatique, Microélectronique et Automatique

Remerciements

Je souhaiterai remercier mes encadrants pour l'aide qu'ils ont pu m'apporter au cours de ce projet. Plus particulièrement, M. Redon pour les différents conseils qu'il a pu m'apporter, son éclaircissement sur les attentes de ce projet et sa disponibilité lors des demandes de matériel ainsi que M. Boé pour l'aide apportée sur la réalisation des cartes électroniques.

Je souhaiterai également remercier M. Flamen pour l'aide qu'il a pu m'apporter sur la réalisation des soudures des cartes électroniques, pour ses précieux conseils et sa disponibilité.

Sommaire

Introduction

En pleine révolution technologique, la robotique est un domaine phare de notre époque. La programmation est enseignée de plus en plus tôt dans les écoles avec des langages simples comme le Python mais l'initiation à la robotique reste un peu plus complexe et présente un coût non négligeable.

Les lycéens ont l'obligation de se procurer une calculatrice graphique à leur arrivée en classe de seconde. Aujourd'hui, il existe des calculatrices graphiques qui permettent la programmation Python, c'est notamment le cas de la calculatrice Numworks. L'objectif de ce projet est alors d'établir une communication entre une calculatrice Numworks et un robot simple et peu coûteux afin de pouvoir le commander en Python à partir de la calculatrice. Cela permettrait aux Lycées de pouvoir initier la robotique aux élèves de manière assez simple, en n'ayant à acheter que le matériel nécessaire à la constitution du robot.

Je vous présenterai dans un premier temps le cahier des charges, le matériel et les attentes de ce projet. Ensuite, j'aborderai la partie réalisation avec notamment la réalisation de cartes électroniques, l'établissement de la communication, le développement d'une API Python pour la calculatrice et enfin, l'établissement d'un exemple d'utilisation de cette réalisation.

I – Présentation du projet

I.1 – Objectif

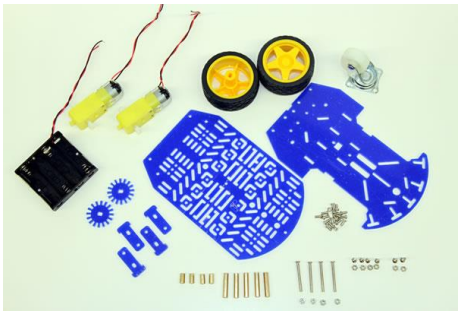
L'objectif de ce projet consistait à établir une communication entre une calculatrice Numworks et un robot classique à base d'Arduino, de capteur de distance et de suiveurs de ligne afin de pouvoir le commander par un script Python depuis la calculatrice.

I.2 – Cahier des charges et matériel

Pour ce projet, les attentes étaient les suivantes :

- Concevoir un robot classique à base d'Arduino Uno, de capteur de distance et de suiveurs de ligne
- Établir une communication entre le robot et la calculatrice NumWorks en tentant de passer le micro-contrôleur en hôte USB et en utilisant le FTDI de l'Arduino
- Développer une API Python permettant de contrôler le robot : fonctions pour contrôler les moteurs, obtenir la valeur du capteur de distance ainsi que celles des suiveurs de ligne
- Développer un programme Python se servant de l'API pour transformer le robot en un suiveur de ligne avec arrêt lors de la rencontre d'un obstacle

Je n'ai pas eu de réel choix de matériel à réaliser pour ce projet étant donné que tout était déjà disponible à Polytech. J'ai donc utilisé :



Kit à assembler Magician chassis dg007

Contrôleur moteur TB6612FNG, C,8, EL

Capteur de distance HC SR04

3 capteurs optique suiveur de ligne QRE1113GR

Arduino Uno

Calculatrice NumWorks

Utilisant des composants montés en surface et afin de réduire la quantité de fils sur le robot, il m'a aussi été demandé de réaliser deux cartes électroniques.

II – Réalisation du projet

II.1 – Réalisation de cartes électroniques

Pour la réalisation des cartes électroniques, j'ai décidé d'utiliser le logiciel Eagle. Mon choix a été motivé par la présence de toutes les footprints des composants utilisés dans ce logiciel ainsi que celle d'un shield vierge pour Arduino Uno. De plus, il est disponible sous Linux contrairement à Altium que nous avons pu utiliser au cours du cursus.

II.1.1 – Shield pour Arduino

Dans un premier temps, je me suis occupé du shield. Ce dernier devait comporter le contrôleur de moteur, le capteur de distance, des headers afin de connecter la batterie, les deux moteurs ainsi qu'un header pour pouvoir lier la deuxième carte électronique comportant les suiveurs de ligne au shield. J'y ai ajouté deux LED afin de faciliter le débogage. Un header pour un capteur de température DHT22 a aussi été ajouté suite à la soutenance intermédiaire durant laquelle un nouvel objectif a été reçu mais qui, par manque de temps, n'a pas été traité au final dans ce projet.

Pour réaliser le shield, je me suis appuyé sur les fiches techniques des composants utilisés. Pour éviter les chutes de tension, des capacités de découplage ont été ajouté entre l'alimentation et la masse.

Après avoir réalisé le schematic, que vous retrouverez en annexe 1, je me suis attelé au routage de la carte. Il a fallu veiller à ce que la taille des pistes du circuit ne soit pas trop petite pour que la carte puisse être imprimée à Polytech. Au moment de l'impression, l'isolation des pistes n'était pas assez grande. En augmentant cette isolation, le plan de masse ne pouvait plus passer entre certaines pastilles et j'ai alors vu apparaître des plans de masse orphelins. Pour palier à ce problème, après avoir tenté de revoir le routage, j'ai finalement ajouté des vias afin de connecter les différentes zones de masse par un fil sur la partie top une fois la carte imprimée. Vous retrouverez le routage en annexe 2.

J'ai ensuite pu graver la carte et réaliser les soudures. Les composants montés en surface ont alors été soudé au four puis il a fallu souder au fer à souder tous les composants traversants.

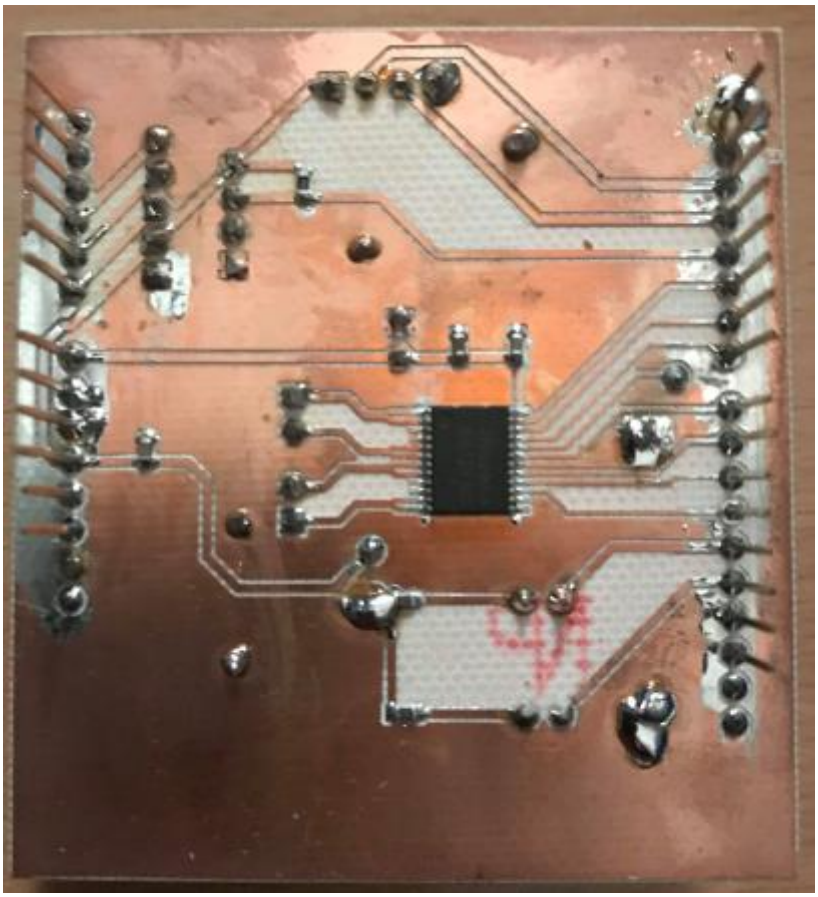


Figure II.1.1.1 : Shield pour Arduino réalisé

Après avoir corrigé quelques courts-circuits apparus avec les soudures, et notamment un sur le pin Rx qui m'empêchait de communiquer avec un autre système, j'ai pu tester la carte et celle-ci fonctionne comme attendue. Je n'ai eu aucun problème de dimensionnement de la carte, puisque la footprint de shield utilisée était déjà correctement dimensionnée pour être adaptée sur un Arduino Uno.

II.1.2 – Carte de suiveurs de ligne

En plus de réaliser un shield, il fallait aussi réaliser une carte pour les suiveurs de ligne puisqu'ils doivent être en dessous du robot pour être utiles. Cette carte a été réalisé beaucoup plus facilement étant donné le peu de composants à y mettre. J'ai choisi d'utiliser 3 capteurs parce que cela est suffisant pour réaliser un bon suivi de ligne et permet aussi de ne pas avoir un code trop long et trop complexe.

Pour établir le schematic, je me suis basé sur la documentation technique des capteurs qui indiquait la valeur des résistances à ajouter pour qu'ils fonctionnent correctement. J'ai ajouté un header afin de pouvoir récupérer les informations renvoyées par les capteurs sur le shield. Vous retrouverez le schematic en annexe 3.

Le routage a lui aussi été particulièrement aisé et l'augmentation de l'isolation des pistes ne m'a pas posé de problème pour cette carte. Vous retrouverez le routage en annexe 4.

Après avoir soudé les capteurs optiques au four et le reste au fer à souder, j'ai pu tester la carte qui a fonctionné du premier coup.

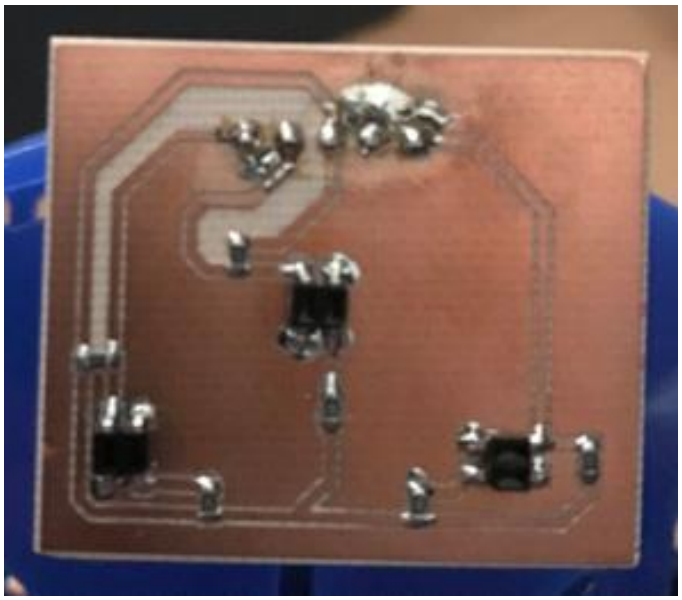


Figure II.1.2.1 : Carte de suiveurs de ligne réalisée

II.2 – Établissement de la communication

II.2.1 – Passage de la Numworks en USB hôte

Afin que l'Arduino du robot et la calculatrice puissent communiquer entre eux, un des objectifs principaux de ce projet était de tenter de passer le microcontrôleur STM32F412VGT6 de la Numworks en mode USB hôte et d'utiliser le circuit adaptateur USB-série de l'Arduino

pour communiquer.

Pour cela, j'ai dans un premier temps étudier la faisabilité. La Numworks étant assez récente, j'ai regardé si elle supportait la spécification USB On-The-Go. L'USB On-The-Go (OTG) est une spécification qui permet à un périphérique USB de se comporter comme un hôte ou un esclave selon la situation. Par exemple, cela permet à une tablette (qui est de base un périphérique esclave) de pouvoir communiquer avec un autre périphérique USB tel qu'une souris, un clavier en agissant comme périphérique hôte puis d'être esclave lorsque connecté à un ordinateur. Cette spécification OTG introduit un 5ème pin, le pin "ID". Lorsque ce dernier est connecté à la masse, le périphérique est défini comme hôte par défaut. Si le pin n'est pas connecté, le périphérique est esclave par défaut.

Le schematic de la Numworks étant disponible sur leur site internet, j'y ai jeté un œil, et plus particulièrement au schematic du port USB :

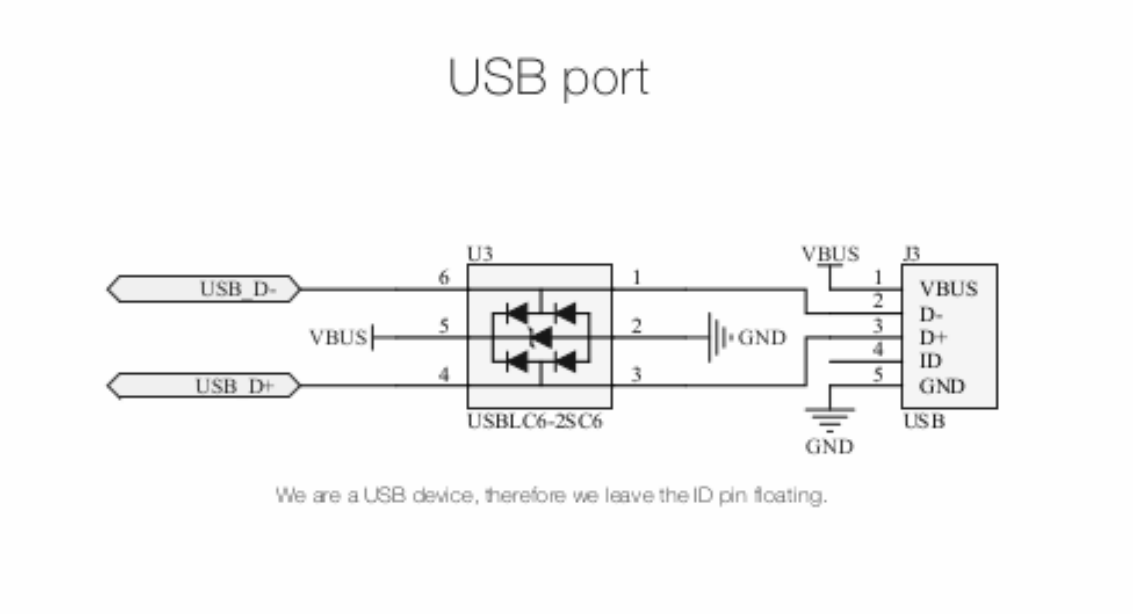


Figure II.2.1.1 : Schematic du port USB de la Numworks

On y observe bien la présence du pin 'ID'. Le port USB de la calculatrice NumWorks n'étant censé être utilisé que pour recharger la calculatrice ou la mettre à jour, la calculatrice est un périphérique esclave. Cela est confirmé par le Schematic de la calculatrice sur lequel on voit que le pin ID de l'USB n'est pas connecté.

Après avoir consulté la documentation du STM32F412VGT6, j'ai pu avoir la confirmation du support de l'OTG. J'ai de plus pu voir plusieurs occurrences du terme 'OTG' dans le code source du système d'exploitation de la Numworks, disponible sur Github que nous avions à modifier.

Je me suis alors lancé dans la modification de l'OS de la calculatrice à l'aide des documentations proposées par STM32. Après avoir modifié l'initialisation de l'USB-OTG du microcontrôleur pour que la calculatrice apparaisse comme un hôte, il fallait encore écrire les fonctions de communication. Avançant dans le flou et ayant encore beaucoup de travail à apporter pour établir la communication, j'ai recherché des bibliothèques déjà faites à utiliser

pour gagner du temps. Je me suis alors rendu compte que STMicroelectronics proposait une bibliothèque pour la communication USB. J'ai été confronté à un problème en essayant de l'intégrer à l'OS de la calculatrice. Numworks ayant implémenté eux-mêmes leurs fonctions de communication et de configuration du périphérique USB, il y avait des collisions à la compilation. En effet, de nombreuses variables sont définies sous le même nom dans le logiciel Numworks et dans les bibliothèques STM mais sont utilisées différemment et d'un côté en C, de l'autre en C++. La seule solution était alors de réécrire la bibliothèque STM en accord avec le code de l'OS Numworks ou alors de modifier le code de la Numworks et notamment renommer tous les registres afin de pouvoir intégrer la bibliothèque STM. Il s'agissait donc d'une quantité de travail importante et fastidieuse.

Au vu du temps restant pour le projet, nous avons pris la décision avec les encadrants lors de la soutenance intermédiaire de s'orienter vers la solution de « secours ». En effet, une modification du logiciel de la calculatrice a déjà été réalisé par zardam pour faire communiquer deux Numworks ensemble. Un UART a été mappé sur le D+ et le D- du port USB. Ainsi, en concevant un câble permettant de connecter la calculatrice à l'Arduino, une communication série est possible avec la modification du logiciel Numworks.

II.2.2 – Modification d'un câble pour permettre la communication

Un câble permettant de relier le port micro-USB de la Numworks au port série de l'Arduino n'existant pas, j'ai dû en réaliser un moi-même. J'ai donc utilisé un câble micro-USB – USB dont j'ai coupé l'extrémité USB. J'ai extrait les 4 fils D+, D-, GND, VCC et les ai dénudés. J'ai soudé 4 fils de connexion afin de pouvoir les connecter plus facilement à l'Arduino. Enfin, j'ai ajouté de la gaine thermo rétractable afin de solidifier le montage et le rendre plus propre.

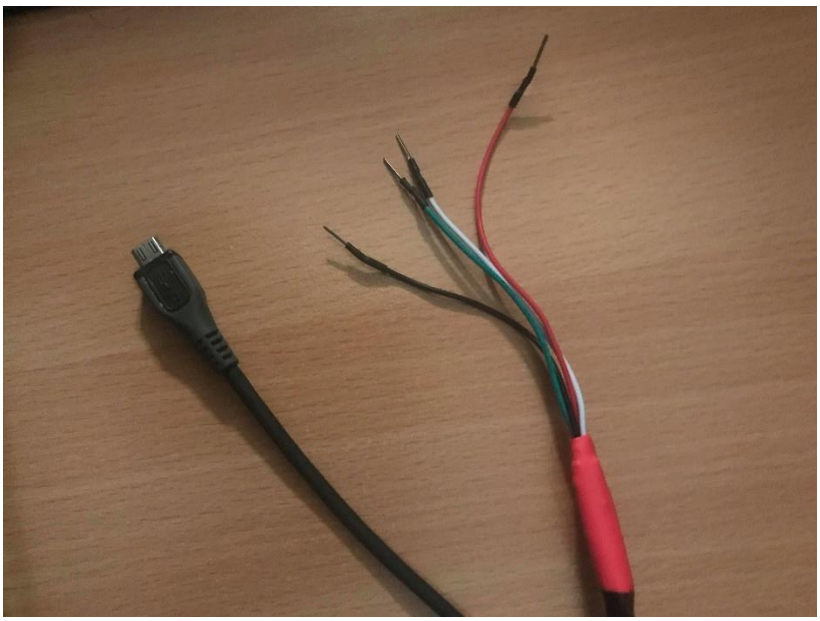


Figure II.2.2.1 : Câble pour la communication Numworks – Arduino

Avec ce câble il est ainsi possible de connecter la Numworks au robot. Pour cela, on connecte :

- Le fil noir (GND) sur GND
- Le fil blanc (Data+) sur le RX
- Le fil vert (Data-) sur le TX
- Le fil rouge (VCC) n'a pas besoin d'être connecté, la Numworks disposant d'une batterie

J'ai alors pu vérifier que la communication était fonctionnelle en utilisant le script exemple implémenté par zardam permettant d'échanger des strings entre les deux périphériques.

II.2.3 – Établissement d'un protocole de communication :

Une fois que la calculatrice et le robot eut été en mesure d'échanger, il a fallu établir un protocole de communication afin qu'ils puissent se comprendre lors de l'échange des messages. La calculatrice doit pouvoir faire comprendre à l'Arduino les différentes commandes qu'elle veut faire exécuter, par exemple, ordonner au robot de tourner sur la gauche. Dans l'autre sens, la calculatrice doit pouvoir comprendre et interpréter les valeurs des capteurs que l'Arduino va lui envoyer. J'ai alors décidé d'utiliser cette forme de trame pour l'envoi de requête de la Numworks vers le robot :

Information	Char de début de trame	Commande	Paramètres	Checksum	Char de fin de trame
Nombre de Char	1	2	3	3	1

La trame débute par le caractère S afin de repérer le début d'une trame, nous avons ensuite la commande définie par deux chiffres qui représentera la commande à exécuter par le robot, les paramètres si besoin, un checksum afin de vérifier s'il n'y a pas eu d'erreur de communication et, finalement, le caractère 'E' qui indique la fin de la trame. Au cours de l'implémentation de la communication, il m'est apparu la nécessité d'ajouter un caractère de séparation ' – ' afin de pouvoir récupérer les différentes informations plus aisément. Ainsi, lorsque nous voudrons par exemple demander au robot d'allumer la LED verte présente sur le shield, la trame « S-01-000-573-E » sera envoyée, « 01 » étant définie comme la commande d'allumage de la LED verte.

Quant à l'envoi de valeurs des capteurs de l'Arduino vers la Numworks, j'utilisais dans un premier temps le même format de trame en plaçant la valeur du capteur en question dans le champ 'Paramètres'. J'envoyais donc une trame par valeur de capteur. Cependant, lors des tests de l'API que je présenterai dans le paragraphe suivant, j'ai été confronté à un problème latence. L'échange d'informations entre la calculatrice et le robot était beaucoup trop long pour pouvoir envisager un suivi de ligne. J'ai alors décidé d'envoyer les valeurs de chaque capteur en une seule trame. Le format de trame de réponse a donc évolué pour devenir le suivant :

Information	Char de début de trame	Capteur de distance	Suiveur de ligne gauche	Suiveur de ligne centre	Suiveur de ligne droit	Checksum	Char de fin de trame
Nombre de Char	1	3	3	3	3	4	1

On retrouve pour la trame de réponse le caractère 'S' de début de trame. Vient ensuite la valeur du capteur de distance, celles des capteurs suiveurs de ligne, le checksum et finalement le caractère 'E' de fin de trame.

Une fois le format des trames défini, j'ai pu procéder à l'implémentation. Du côté de l'Arduino, on attend de recevoir une trame sur le port série. Lorsque qu'une trame est reçue, on la stocke puis on réalise un découpage pour isoler de la trame la commande à réaliser ainsi que le checksum. On vérifie ensuite qu'il n'y a pas eu d'erreur de communication en recalculant le checksum et en le comparant à celui reçu. Si le checksum est bon, on exécute la commande voulue. Pour la réponse, lorsque qu'une demande de mesure des capteurs a été reçue, on récupère la valeur de chaque capteur, on forme la trame à envoyer selon le modèle présenté précédemment et en prenant soin de calculer le checksum puis on envoie la trame sur le port série. Pour les commandes autres que celles demandant les valeurs des capteurs, par exemple, demander au robot d'avancer, je pensais au premier abord qu'il n'y avait pas besoin de réponse. Je me suis cependant rendu compte qu'à partir du moment où l'on commençait à échanger plusieurs trames à la suite, un problème de synchronisation survenait. La commande n'avait pas encore été exécuté sur le robot que la Numworks avait déjà envoyé la suivante. Pour régler ce problème, lorsque le robot reçoit une commande ne nécessitant pas de réponse, il envoie tout de même le message « OK » pour prévenir la Numworks que la commande a bien été exécuté. Ainsi, du côté de la Numworks, on attend d'avoir reçu « OK » pour passer à la commande suivante.

L'implémentation du côté de la Numworks a été un peu plus compliqué à mettre en place. La Numworks propose un environnement Python grâce à l'embarquement de Micropython. Afin de mettre en place la communication et l'API, il a fallu développer un nouveau module Micropython. Pour cela, le développement se fait en C mais avec des types et structures bien particuliers et spécifiques à Micropython et on ne trouve pas énormément de documentation à ce propos sur internet. Pour l'écriture et la lecture du port série, l'auteur de la modification du logiciel Numworks a mis en place deux fonctions :

```
mp_obj_t writeLine(mp_obj_t a)
```

et

```
mp_obj_t readLine()
```

Après avoir compris l'utilisation des types utilisés par Micropython, j'ai pu utiliser ces fonctions pour implémenter la communication. J'ai alors implémenté une fonction de formation de trame prenant en paramètre la commande à envoyer et calculant le checksum. Une fois la trame formée, elle est envoyée grâce à la fonction implémentée par zardam writeLine(mp_obj_t a). Pour la réception, on procède de la même manière que sur l'Arduino, on attend la réception du trame sur le port série grâce à la fonction readLine(), on effectue un découpage en stockant séparément chaque valeur de capteur ainsi que le checksum. On vérifie que le checksum est correct en le recalculant et en le comparant à celui reçu. Si ce n'est pas le cas, on renvoie la requête à l'Arduino.

Lorsque l'on envoie une demande de valeurs de capteurs, on attend de la recevoir avant de passer à la requête suivante. Pour les autres types de requête, on attend de recevoir 'OK' comme expliqué précédemment pour pouvoir passer à la suite.

Il y a ainsi deux scénarios de communication :

- Les fonctions nécessitant une réponse « OK ». Exemple : l'utilisateur lance la fonction de l'API permettant d'allumer la LED verte. La trame « S-01-000-573-E » est formée sur la Numworks puis envoyée au robot. La trame est reçue par l'Arduino, découpée, le checksum est vérifié puis la commande demandée est exécutée en faisant appel à la fonction correspondante. Une fois terminée, la trame « OK » est envoyée à la Numworks. La fonction exécutée sur la Numworks qui attendait la réponse « OK » rend la main.
- La fonction demandant les valeurs de capteurs. L'utilisateur lance la fonction de l'API measureSensors() permettant de récupérer les valeurs des capteurs. La trame « S-04-000-576-E » est envoyée à l'Arduino, découpée, le checksum est vérifié puis la commande demandée est exécutée en faisant appel à la fonction responsable de chaque capteur. La trame de réponse : « S-xxx-yyy-zzz-wwww-E » contenant les valeurs des capteurs est alors envoyée à la Numworks qui rend la main.

II.3 – Mise à disposition d'une API Python :

Afin de permettre à l'utilisateur de commander le robot à partir de la calculatrice Numworks, j'ai mis en place une API python simple que j'ai nommé « robot ». L'utilisateur peut alors accéder à toutes les fonctions de commande du robot après l'avoir importé dans son script python par la ligne :

```
import robot
```

II.3.1 – Développement de l'API :

Comme mentionné précédemment, afin de mettre à disposition à l'utilisateur une API python de commande du robot, il a fallu créer un nouveau module Micropython. Pour cela, plusieurs fichiers du code source Micropython sont à modifier. En effet, il faut préciser à Micropython ou se trouve le module développé, lui préciser que ce nouveau module est utilisable par l'utilisateur et que le mot « robot » fait désormais référence à ce module.

Après avoir réalisé les modifications nécessaires à la création du nouveau module, j'ai pu implémenter l'API. L'API devait donc permettre de contrôler les moteurs du robot et récupérer les valeurs des différents capteurs. Ayant deux LED, j'ai fait en sorte que l'API puisse permettre aussi à l'utilisateur de commander ces dernières. Les fonctions développées et proposées à l'utilisateur sont les suivantes :

- void ledRouge() : Demande au robot d'allumer la LED rouge. Une fois la commande exécutée par le robot, un « OK » de retour est envoyé. Cette fonction attend que la commande ait bien été exécutée avant de rendre la main.

- void ledVerte() : Demande au robot d'allumer la LED verte. Une fois la commande exécutée par le robot, un « OK » de retour est envoyé. Cette fonction attend que la commande ait bien été exécutée avant de rendre la main.

- void moveForward() : Demande au robot d'avancer. Une fois la commande exécutée par le robot, un « OK » de retour est envoyé. Cette fonction attend que la commande ait bien été exécutée avant de rendre la main.

- void moveBackward() : Demande au robot de reculer. Une fois la commande exécutée par le robot, un « OK » de retour est envoyé. Cette fonction attend que la commande ait bien été exécutée avant de rendre la main.

- void moveRight() : Demande au robot de tourner à droite. Une fois la commande exécutée par le robot, un « OK » de retour est envoyé. Cette fonction attend que la commande ait bien été exécutée avant de rendre la main.

- void moveLeft() : Demande au robot de tourner à gauche. Une fois la commande exécutée par le robot, un « OK » de retour est envoyé. Cette fonction attend que la commande ait bien été exécutée avant de rendre la main.

- void stop() : Demande au robot de s'arrêter. Une fois la commande exécutée par le robot, un « OK » de retour est envoyé. Cette fonction attend que la commande ait bien été exécutée avant de rendre la main.

- void measureSensors() : Demande au robot d'envoyer les valeurs du capteur de distance et des trois suiveurs de ligne. La Numworks attend alors que l'Arduino lui envoie la trame contenant les différentes valeurs. Les valeurs sont ensuite stockées.

- int getDistanceSensorValue() : retourne la valeur du capteur de distance en mm, c'est-à-dire, la distance à laquelle se trouve le robot d'un obstacle. Afin que le nombre de caractères représentant la valeur ne varie pas, cette fonction a été implémentée pour renvoyer une distance comprise entre 100mm et 999mm. Pour qu'elle soit utilisable, il faut au préalable avoir fait un appel à la fonction measureSensors() afin d'être en possession des valeurs car cette fonction n'effectue aucun échange avec le robot, elle ne fait que lire une valeur stockée.

- int getLineFollowerSensorGValue() : retourne la valeur du suiveur de ligne gauche. Afin que le nombre de caractères représentant la valeur ne varie pas, cette fonction a été implémentée pour retourner une valeur comprise entre 100 et 999. Pour qu'elle soit utilisable, il faut au préalable avoir fait un appel à la fonction measureSensors() afin d'être en possession des valeurs car cette fonction n'effectue aucun échange avec le robot, elle ne fait que lire une valeur stockée.

- int getLineFollowerSensorCValue() : retourne la valeur du suiveur de ligne centre. Afin que le nombre de caractères représentant la valeur ne varie pas, cette fonction a été implémentée pour retourner une valeur comprise entre 100 et 999. Pour qu'elle soit utilisable, il faut au préalable avoir fait un appel à la fonction measureSensors() afin d'être en possession des valeurs car cette fonction n'effectue aucun échange avec le robot, elle ne fait que lire une valeur stockée.

- int getLineFollowerSensorDValue() : retourne la valeur du suiveur de ligne droit. Afin que le nombre de caractères représentant la valeur ne varie pas, cette fonction a été implémentée pour retourner une valeur comprise entre 100 et 999. Pour qu'elle soit utilisable, il faut au préalable avoir fait un appel à la fonction measureSensors() afin d'être en possession des valeurs car cette fonction n'effectue aucun échange avec le robot, elle ne fait que lire une valeur stockée.

Les seuls points qui ont pu poser problèmes lors du développement de cette API ont encore une fois été l'utilisation correcte des types Micropython. Du côté de l'Arduino, le développement a été relativement aisé puisque les fonctions d'exécutions de commande à implémenter étaient assez communes.

Une fois la communication fonctionnelle entre l'Arduino et la Numworks, j'ai pu tester le bon fonctionnement de l'API. Pour cela, plutôt que de taper les instructions à la calculatrice à chaque test avec le module Python, j'ai intégré un script de test dans le code source de Micropython dans l'OS. Une fois le script mis en place, je pouvais le retrouver sur la calculatrice et l'utiliser.

II.3.2 – Exemple d'utilisation de l'API :

La dernière partie de ce projet consistait à proposer un script d'exemple d'utilisation de l'API développée. Ce script devait permettre de commander le robot pour qu'il suive une ligne de manière autonome et s'arrête en cas de rencontre d'un obstacle.

Pour cela, j'ai dans un premier temps implémenté l'algorithme sur l'Arduino seule afin d'être sûr de son fonctionnement et d'éviter de chercher des erreurs là où il n'y en a pas. Je me suis vite rendu compte qu'un simple algorithme comme le suivant ne permettrait pas le suivi de ligne :

```
Tant que 1 :
    Si obstacle :
      s'arrêter
    Sinon si capteur centre sur la ligne :
      avancer tout droit
    Sinon si capteur gauche sur la ligne :
      tourner à gauche
    Sinon si capteur droit sur la ligne :
      tourner à droite
  Fin tant que
```

En effet, un algorithme aussi simple ne me permettait pas de terminer des parcours présentant des angles droits. Lorsque le robot arrive sur un angle droit, la plupart du temps,

le capteur gauche et le capteur droit sont simultanément sur la ligne. Pour que le robot sache dans quel sens tourner, j'ai pensé à mémoriser le capteur qui était le premier sur la ligne. Ainsi, lorsque le robot rencontre l'angle droit, il doit tourner à gauche si le dernier capteur à être passer sur la ligne était celui de gauche et inversement. Après avoir implémenté cela, le robot à commencer à prendre les virages correctement mais j'ai vu un nouveau problème apparaître. Lorsqu'il prenait un angle droit, sa vitesse faisait qu'il dépassait la ligne dans le virage en tournant et n'ayant plus aucun capteur sur la ligne, il ne savait plus quoi faire. En mémorisant la dernière action réalisée, on peut lui indiquer de continuer celle-ci si plus aucun capteur ne se trouve sur la ligne. Ainsi, l'algorithme suivant est fonctionnel :

```
Tant que 1 :
    Si obstacle :
      s'arrêter
    Sinon si capteur centre sur la ligne ET capteur G et D pas sur la
ligne :
      avancer tout droit
    Sinon si capteur G sur la ligne ET capteur D pas sur la ligne :
      tourner à gauche
      derniereAction <- gauche
    Sinon si capteur D sur la ligne ET capteur G pas sur la ligne:
      tourner à droite
      dernieAction <- droite
    Sinon si capteur G ET capteur D sur la ligne :
      Si dernieAction == droite :
        Tourner à gauche
        dernieAction <- gauche
      Sinon si dernieAction == gauche :
        Tourner à droite
        dernieAction <- droite
    Sinon si capteur G et D et C pas sur la ligne :
      Si dernieAction == droite :
        Tourner à droite
      Sinon si dernieAction == gauche :
        Tourner à gauche
  Fin tant que
```

Une fois l'algorithme de suivi de ligne fonctionnel sur l'Arduino, j'ai pu le porter sur la Numworks. J'ai cependant été confronté à un problème d'échange de données entre la Numworks et le robot trop longs comme mentionné précédemment. Après avoir changé la façon de récupérer les valeurs des capteurs qui n'a pas vraiment arrangé le problème, j'ai fait quelques tests et me suis rendu compte que la fonction de lecture de trame sur l'Arduino introduisait une latence énorme. Après l'avoir réimplémenté, les échanges étaient beaucoup plus rapides et le suivi de ligne a été possible.

Le script implémenté sur la Numworks est alors le suivant :

```
constexpr ScriptTemplate suiviLigneScriptTemplate("suiviLigne.py", R"(import robot
def go():
  right = 0
  left = 0
  seuil = 900
  while(1):
    robot.measureSensors()
    if (robot.getDistanceSensorValue() == 100):
      robot.stop()
      robot.ledRouge()
      return
    elif ((robot.getLineFollowerSensorCValue() > seuil) & (robot.getLineFollowerSensorGValue() < seuil) & (robot.getLineFollowerSensorDValue() < seuil)):
      robot.moveForward()
    elif( (robot.getLineFollowerSensorGValue() > seuil) & (robot.getLineFollowerSensorDValue() < seuil)):
      robot.moveLeft()
      left = 1
      right = 0
    elif( (robot.getLineFollowerSensorDValue() > seuil) & (robot.getLineFollowerSensorGValue()<seuil)):
      robot.moveRight()
      right = 1
      left = 0
    elif ((robot.getLineFollowerSensorGValue() > seuil) & (robot.getLineFollowerSensorDValue()> seuil)):
      if (right ==1):
        robot.moveLeft()
        left = 1
        right = 0
      elif (left ==1):
        robot.moveRight()
        right = 1
        left = 0
    elif ((robot.getLineFollowerSensorGValue() <seuil) & (robot.getLineFollowerSensorCValue()< seuil) & (robot.getLineFollowerSensorDValue()<seuil)):
      if (right ==1):
        robot.moveRight()
        left = 0
        right = 1
      elif (left ==1):
        robot.moveLeft()
        right = 0
        left = 1)");
```

Figure II.3.2.1 : Script de suivi de ligne

Ce script est disponible dans le module Python de la Numworks sous le nom « suiviLigne.py »

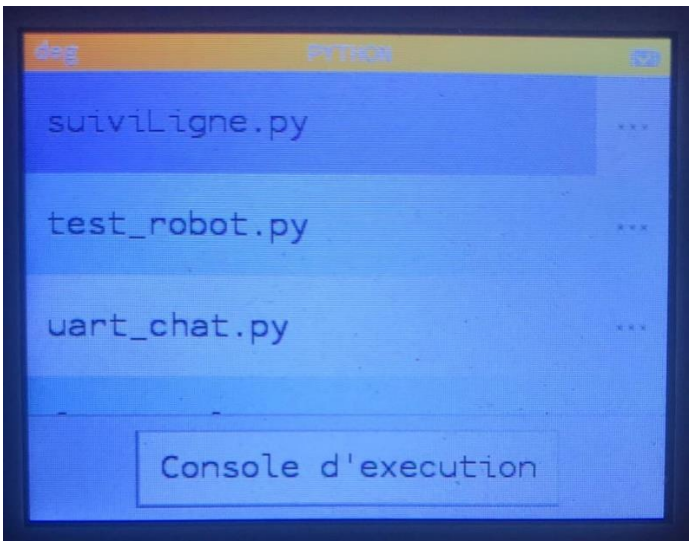


Figure II.3.2.2 : Présence du script sur la Numworks

Afin de fonctionner correctement, le robot doit être placé sur la ligne avant le lancement du script. Pour lancer la commande, il faut exécuter la fonction go() présente dans ce script. La fonction ne rendra alors la main que lorsque le robot aura détecté un obstacle à 10 cm de lui.

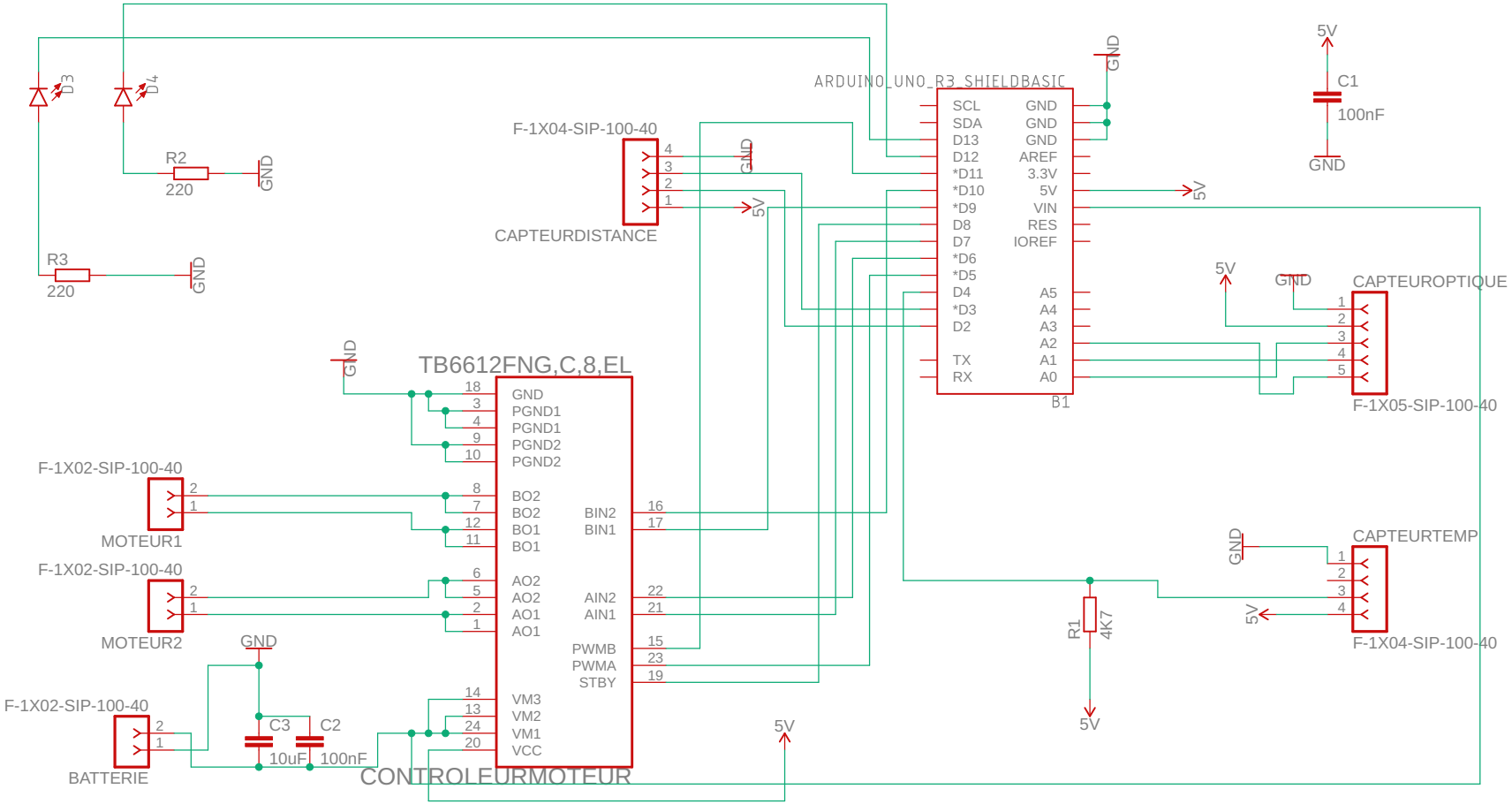
Conclusion

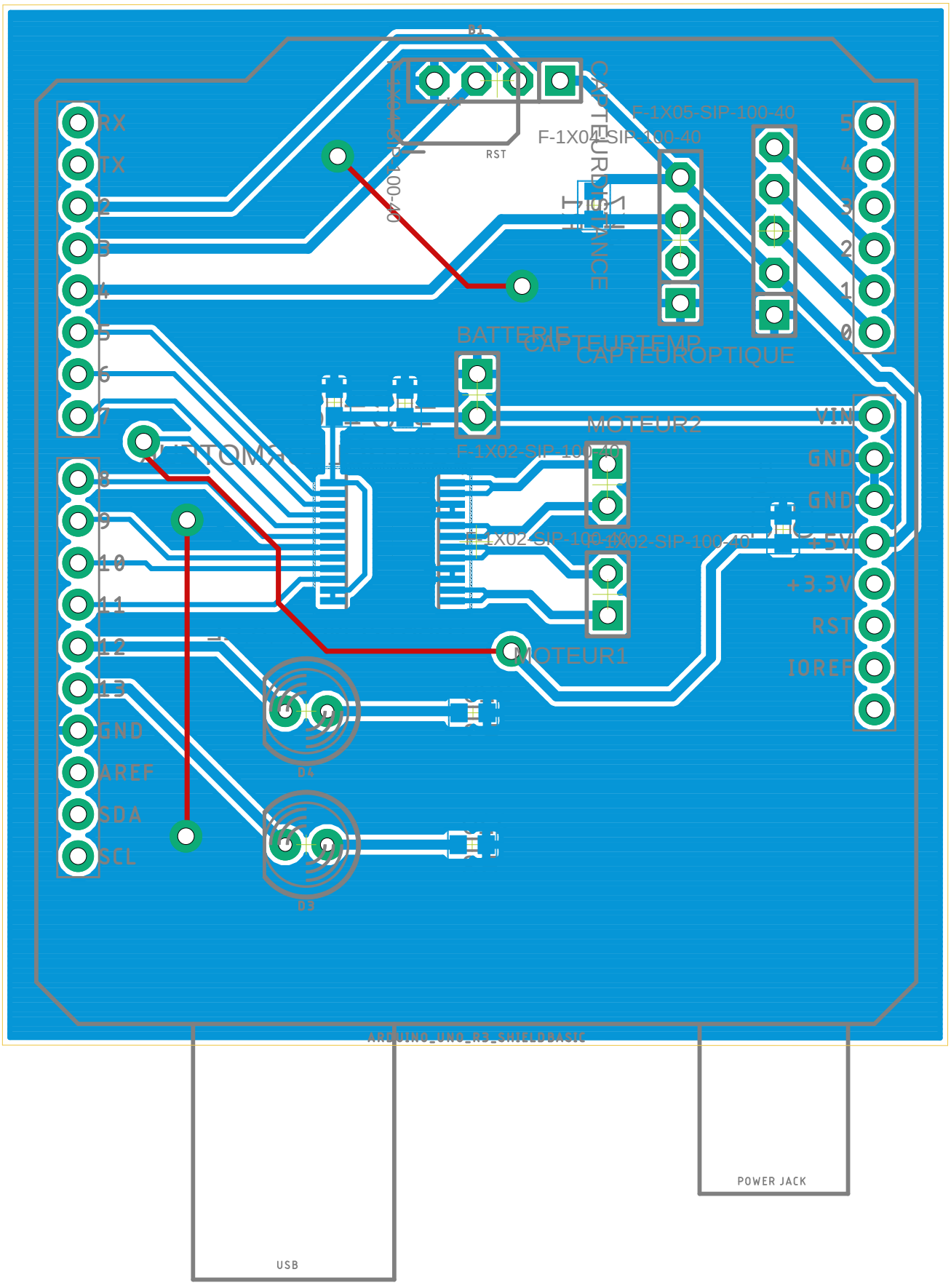
Bien que la modification du logiciel Numworks pour faire de la calculatrice un hôte USB était possible, elle demandait une quantité de travail importante et fastidieuse qui n'aurait pas permis de terminer le projet. En se tournant vers la solution de « secours » je suis parvenu à établir une communication entre la calculatrice et le robot et ce projet aboutit sur une API fonctionnelle de commande du robot restant assez simple d'utilisation et étant destiné à des lycéens. Le script développé permet à l'utilisateur de voir comment utiliser l'API tout en permettant la commande du robot pour un suivi de ligne autonome avec arrêt lors de la rencontre d'un obstacle.

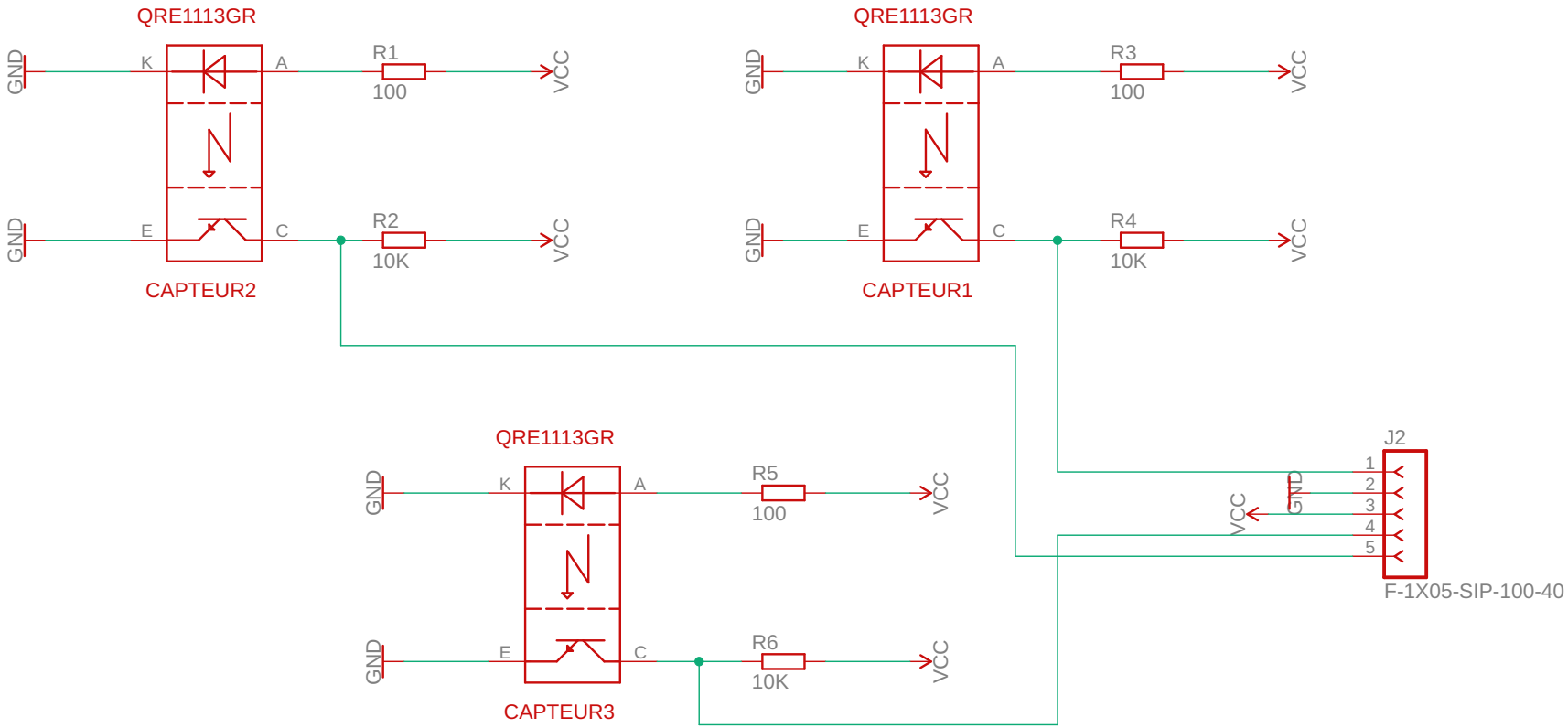
Au final, ce projet m'aura permis d'accroitre mes capacités de recherche, mon autonomie ainsi que la gestion de projet. Il m'a aussi permis de consolider mes connaissances sur l'environnement Arduino, n'ayant pas fait PEIP, je n'avais aucune expérience outre celle du projet de 3ème année.

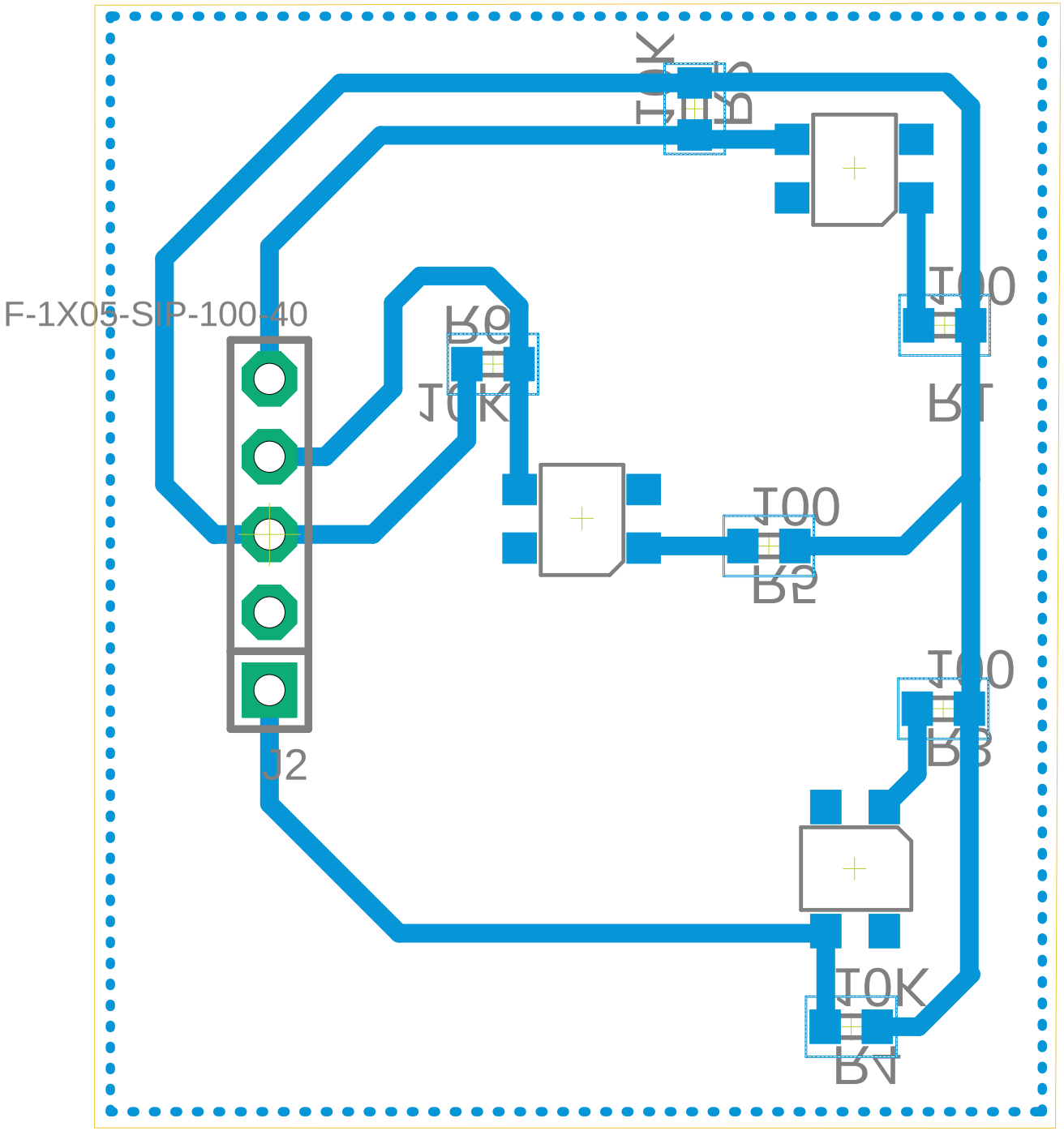
Bibliographie

- https://zardam.github.io/post/numworks-uart-over-usb/
- https://github.com/zardam/epsilon/tree/uart_over_usb
- https://github.com/numworks/epsilon
- https://www.numworks.com/
- https://www.st.com/content/ccc/resource/technical/document/user_manual/b8/5a/28/c2/cf/b6/47/d6/DM00105256.pdf/files/DM00105256.pdf/jcr:content/translations/en.DM00105256.pdf
- https://www.st.com/content/ccc/resource/technical/document/user_manual/1c/6b/06/e6/19/6c/46/bf/CD00289278.pdf/files/CD00289278.pdf/jcr:content/translations/en.CD00289278.pdf
- https://www.st.com/content/ccc/resource/technical/document/reference_manual/group0/4f/7b/2b/bd/04/b3/49/25/DM00180369/files/DM00180369.pdf/jcr:content/translations/en.DM00180369.pdf
- https://peip-ima.plil.fr/mediawiki/index.php/BE_2017-2018
- http://aaroneiche.com/2010/06/24/a-beginners-guide-to-making-an-arduino-shield-pcb/
- https://www.open-electronics.org/how-to-make-an-arduino-shield-with-eagle-cad-tutorial/
- https://docs-emea.rs-online.com/webdocs/10e2/0900766b810e25b7.pdf
- https://www.sparkfun.com/datasheets/Robotics/TB6612FNG.pdf
- https://forum.micropython.org/viewtopic.php?t=776
- https://micropython-dev-docs.readthedocs.io/en/latest/adding-module.html
- https://www.snapeda.com/parts/QRE1113GR/Fairchild%20Semiconductor/view-part/
- https://www.snapeda.com/parts/POLOLU-713/Pololu/view-part/
- http://www.diymodules.org/eagle-show-object?type=dm&file=diy-modules.lbr&device=ULTRASONIC-HC-SR04
- https://odpf.org/images/archives_docs/17eme/memoires/gr-5/memoire.pdf
- https://www.techiedelight.com/implement-itoa-function-in-c/


D3
D4
R2
220
GND
R3
220
GND
F-1X04-SIP-100-40
CAPTEURDISTANCE
GND
5V
ARDUINO_UNO_R3_SHIELDBASIC
SCL
SDA
D13
D12
*D11
*D10
*D9
D8
D7
*D6
*D5
D4
*D3
D2
TX
RX
GND
GND
AREF
3.3V
5V
VIN
RES
IOREF
A5
A4
A3
A2
A1
A0
B1
5V
C1
100nF
GND
5V
GND
CAPTEUROPTIQUE
F-1X05-SIP-100-40
GND
TB6612FNG,C,8,EL
GND
PGND1
PGND1
PGND2
PGND2
BO2
BO2
BO1
BO1
AO2
AO2
AO1
AO1
BIN2
BIN1
AIN2
AIN1
PWMB
PWMA
STBY
VM3
VM2
VM1
VCC
F-1X02-SIP-100-40
MOTEUR1
F-1X02-SIP-100-40
MOTEUR2
F-1X02-SIP-100-40
BATTERIE
GND
C3
10uF
C2
100nF
CONTROLEURMOTEUR
5V
R1
4K7
5V
GND
CAPTEURTEMP
5V
F-1X04-SIP-100-40


RX
TX
2
3
4
5
6
7
8
9
10
11
12
13
GND
AREF
SDA
SCL
RST
F-1X04-SIP-100-40
F-1X05-SIP-100-40
CAPTEURDISTANCE
BATTERIE
CAPTEURTEMP
CAPTEUROPTIQUE
MOTEUR2
F-1X02-SIP-100-40
MOTEUR1
D4
D3
VIN
GND
GND
+5V
+3.3V
RST
IOREF
ARDUINO_UNO_R3_SHIELDBASIC
USB
POWER JACK


QRE1113GR
GND
K
A
R1
100
VCC
GND
E
C
R2
10K
VCC
CAPTEUR2
QRE1113GR
GND
K
A
R3
100
VCC
GND
E
C
R4
10K
VCC
CAPTEUR1
QRE1113GR
GND
K
A
R5
100
VCC
GND
E
C
R6
10K
VCC
CAPTEUR3
J2
1
2
3
4
5
VCC
GND
F-1X05-SIP-100-40


F-1X05-SIP-100-40
J2