

Rapport de Stage

Projet Le Cube

Stagiaire : Théo DHUIEGE

Tuteur entreprise : Sébastien CARRIERE

Tuteur enseignant : Pascal HUSSAUD

Période du 08/04/2024 au 14/06/2024

Sommaire

Liste des images et des tableaux	3
Présentation du laboratoire des technologies innovantes	4
Genèse du projet	4-5
Tâches réalisées durant le stage	5
Cahier des charges	5-6
Première version	
Composants de la maquette	7-8
Communication avec les composants	8-10
Deuxième version	
Ecriture en morse et affichage à l'écran	11-12
Programmation par PlatformIO	13
Troisième version	
Changement de la forme et de carte Arduino	14-15
Intelligence artificielle et reconnaissance des mouvements	15-17
Nouveau moyen de communication	18
Communication Bluetooth	18-19
Présentation du projet au douzième Challenge Handicap et Technologies	20
Conclusion	21
Annexes	22-34

Liste des images et des tableaux :

Figure	Page	Libellé
1	7	Maquette de prototypage basée sur une Arduino Nano classique.
2	8	Cube vu de l'extérieur. La face 3 se situe de l'autre côté de la face 1.
3	8	Cube vu de l'intérieur avec le capteur MPR121.
4	8	Carte du capteur MPR121
5	9	Schéma de liaison de maîtres et d'esclaves I ² C.
6	9	Trame d'une communication I ² C.
7	10	Schéma de liaison de maîtres et d'esclaves SPI.
8	11	Message d'erreur signifiant l'absence du MPR121.
9	11	Ecran du système au démarrage.
10	12	Utilisation du système.
11	12	Réaction du système face à un code inconnu.
12	13	Contenu d'un fichier platformio.ini.
13	13	Contenu d'un dossier de projet sur PlatformIO.
14	14	Vue externe de la demi-sphère supérieure avec les contacts.
15	14	Vue interne de la demi-sphère supérieure avec le MPR121 intégré.
16	14	Carte sur laquelle repose l'Arduino avec les connecteurs et le bouton.
17	15	Début du fichier csv contenant la description des colonnes et les premières mesures du premier échantillon.
18	16	Taux d'erreur de l'IA au fur et à mesure des étapes de recherche. Plus cette valeur est petite, plus l'IA devient performante.
19	16	Erreur moyenne absolue de l'IA au fur et à mesure des étapes de recherche. Plus cette valeur est petite, plus l'IA devient performante.
20	17	Terminal série de l'Arduino, montrant les mouvements détectés.
21	18	Tableau des caractères du nouveau moyen de communication
22	19	Réaction des mouvements de la sphère vue sur le terminal Bluetooth.
23	19	Réaction des contacts de la sphère vue sur le terminal Bluetooth.
24	22	Communication avec la chercheuse de l'université Paris 8 rencontré au challenge.
25	23	Schéma électronique de la première et de la deuxième version.
26	23	Schéma électronique de la troisième version.

Présentation du laboratoire des technologies innovantes

Crée par un groupement de deux équipes de recherche de l'Université de Picardie Jules Verne en 2004, le Laboratoire des Technologies Innovantes est une équipe de chercheurs et de chercheuses spécialisée en l'ingénierie des matériaux et la modélisation de systèmes complexes.

Composé d'une cinquantaine de permanents et d'une quarantaine d'enseignants-chercheurs, leurs activités de recherches s'articulent autour de l'utilisation efficace des ressources énergétiques pour un développement durable.

Le laboratoire s'étend depuis 2012 sur quatre thématiques :

- Matériaux et Efficacité Energétique,
- Mécanique et Ingénierie des Matériaux,
- Systèmes Intelligents,
- Energie Electrique et Systèmes Associés.

Les chercheurs et chercheuses du laboratoire proviennent et travaillent dans divers campus de l'Université de Picardie tels que Amiens, Saint-Quentin ou encore Soissons/Cuffies, où j'ai été personnellement affilié. Les travaux en cours concernent principalement les matériaux nouveaux et l'optimisation énergétique dans les domaines de la construction et des transports.

Genèse du projet

Une personne est venue demander notre aide pour développer un moyen de communiquer facilement avec sa fille, qu'on appellera Aurélie dans ce rapport.

Aurélie a eu il y a quelques années un grave accident domestique ayant entraîné un traumatisme crânien, provoquant une période de coma de 2 ans. A son réveil, Aurélie est tétraplégique, sauf à la main gauche, où elle peut effectuer de très légers mouvements.

Avant son accident, Aurélie a appris à communiquer en utilisant le langage morse, une obligation lorsque l'on passe le permis bateau, ce qu'elle faisait. Père et fille communiquent actuellement en morse en utilisant un crayon, représentant le trait, et un jeton, représentant le point. Aurélie tient ces deux objets dans sa main gauche et jette l'élément qu'elle ne veut pas utiliser. Un tel système est fonctionnel mais lent : en effet, aurélie n'a plus conscience du temps que peut prendre les actions qu'elle souhaite faire et un simple mot de six à huit lettres peut prendre plusieurs minutes voire plusieurs heures à écrire. De plus, son père doit ramasser puis redonner l'objet que sa fille a jeté et décoder ce qu'elle a dit. Enfin, les risques d'erreurs et de confusion, car si Aurélie se souvient du morse, elle confond certaines lettres, ralentissent d'autant plus la discussion.

Notre mission est donc de développer un système permettant à Aurélie de communiquer en morse plus rapidement et qu'une traduction en textuel soit fait en direct pour lui permettre de parler avec tout le monde.

Tâches réalisées durant le stage.

Mes tâches dans le cadre de mon stage sont de réaliser les programmes de la deuxième et de la troisième version du projet. Ces tâches ont été pour certaines sans difficulté car ce sont des domaines que je connais mais pour d'autre, notamment dans le cadre de la troisième version, j'ai eu besoin de découvrir de nouvelles notions car elles sont globalement inconnues pour moi du point de vue de la programmation.

Une autre tâche a été de présenter le projet et son histoire lors du douzième Challenge Handicap et Technologie qui a eu lieu à l'Ecole Nationale d'Ingénieurs de Metz les 23 et 24 mai 2024.

Cahier des charges

Première version :

- Système basé sur une carte Arduino Nano.
- Cube posé et retenu sur la main via une bande de velcro pour qu'il ne tombe pas même s'il n'est pas tenu par les doigts.
- Utilisation de deux des quatre contacts du cube afin d'entraîner Aurélie à bouger ses doigts.
- Réactions auditives et visuelles des contacts via un haut-parleur, un écran et des LED.

Deuxième version :

- Système et cube identique à la première version.
- Discussion en morse, avec les 4 contacts : point (.), trait (-), caractère suivant et effacer. Les contacts sont détectés par un capteur MPR121.
- Affichage du caractère en cours d'écriture et de la phrase en cours d'écriture à l'écran.
- Réaction du haut-parleur et des LED lorsque l'on appuie sur le contact du trait ou du point.

Troisième version :

- Système basé sur une carte Arduino Nano 33 BLE Sense Rev2.
- Sphère posée et retenue sur la main via une bande de velcro pour qu'elle ne tombe pas même si elle n'est pas tenue par les doigts. La sphère dispose de cinq contacts.
- Discussion inspirée des téléphones à touches : trois contacts à toucher une ou plusieurs fois pour sélectionner une lettre, une quatrième pour passer à la lettre suivante et une dernière pour effectuer un retour arrière.
- Ajout d'une détection du mouvement déterminée par les l'IMU interne de l'Arduino. La reconnaissance de ces mouvements est effectuée par le service d'intelligence artificielle Google TensorFlow.
- Affichage du texte écrit par l'utilisateur par une application du type terminal Bluetooth et qui communiquera avec l'Arduino via du Bluetooth Low Energy.

Première version

Avant mon arrivée en stage, la maquette ci-dessous a été construite par Pascal Hussaud et Christophe Malinowski en vue de faire les premiers essais. Cette maquette contient le capteur de contact MPR121, un écran de type ST7789, deux matrices de LED alimentées par une alimentation extérieure et un JR6001, une carte permettant de lire n'importe quel fichier audio qu'on lui aura transféré dans son stockage interne sur un haut-parleur piézoélectrique. La première version du programme a été développée avant mon stage et avait pour but de stimuler et de rééduquer Aurélie.

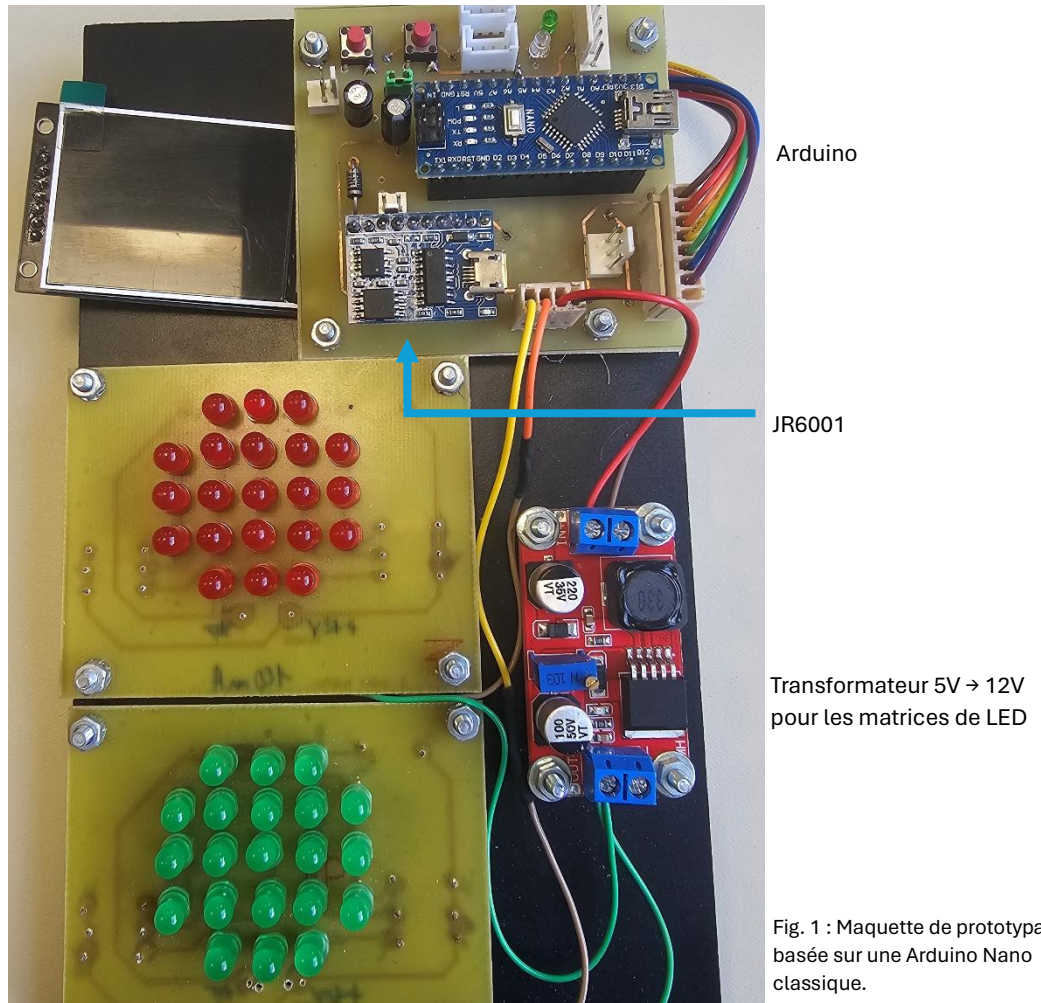


Fig. 1 : Maquette de prototypage basée sur une Arduino Nano classique.

Le cube dispose de quatre surfaces tactiles. La surface 1 est pour le pouce, la surface 2 est pour les quatre autres doigts, la surface 3 est contre le pouce et la surface 0 est au-dessus. Lorsqu'Aurélien touche la surface 1 ou 2, un son est émis, une des deux matrices s'allume et l'écran affiche un point ou un trait pour montrer une réaction face à ses gestes.

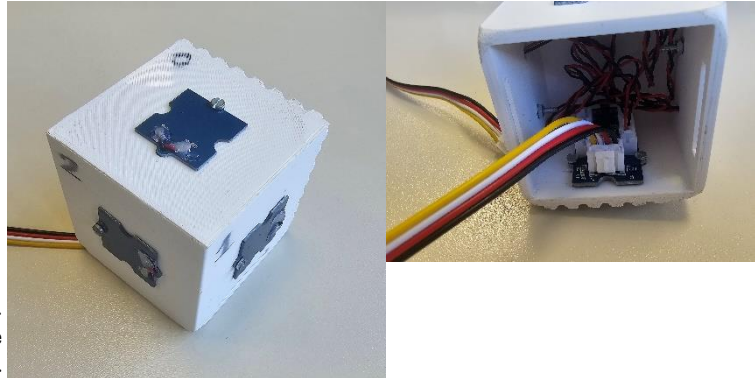


Fig. 2 : Cube vu de l'extérieur.
La face 3 se situe de l'autre côté de la face 1.

Fig. 3 : Cube vu de l'intérieur avec le capteur MPR121.

Le capteur MPR121 est une carte comprenant 12 contacts permettant de détecter si l'on touche ces contacts. Le capteur communique en I²C et envoie un nombre compris entre 0 et 4095, correspondant aux capteurs touchés. Chaque capteur vaut une puissance de 2 : le contact 0 vaut 1, le contact 1 vaut 2, jusqu'au contact 11 valant 2048. Si le capteur renvoie.

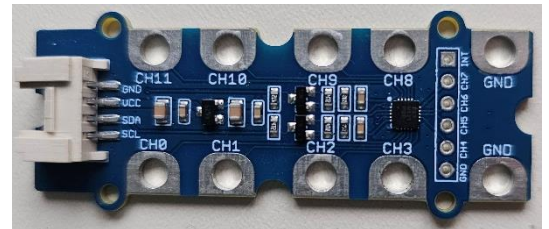


Fig. 4 : Carte du capteur MPR121.

L'écran utilisé ici est de la marque Adafruit et basé sur un contrôleur ST7789. Il fait 320 par 240 pixels, capable d'afficher de la couleur. Cet écran communique avec l'Arduino via une communication SPI.

Le JR6001 est un circuit disposant d'une mémoire interne se comportant comme un dispositif de stockage. En le connectant à un ordinateur, il est possible d'y stocker des fichiers sonores au format MP3 ou WAV. L'Arduino est capable de contrôler ce circuit en lui envoyant des messages via une communication série asynchrone par deux broches Tx et Rx. Ensuite, deux broches du JR6001 lui permettent de transférer le signal directement sur un haut-parleur et deux autres lui permettent d'envoyer un signal stéréo vers une connectique ou un circuit d'amplification.

Communication avec les composants

Le JR6001 utilise une communication série via deux broches de l'Arduino. Cette communication consiste en deux broches permettant au JR6001 et à l'Arduino d'envoyer des données à l'autre composant. Une nécessité importante à prévoir pour que la communication fonctionne correctement est que les réglages, notamment de vitesse de communication, exprimé en bauds ou bits par seconde, soient identiques entre les deux appareils.

Ce message envoyé est sous cette forme : "AX : 0" avec AX : Une lettre 'A' ou 'B' suivi d'une valeur hexadécimal, permettant de choisir l'action à effectuer parmi une liste donnée dans la documentation du module, et 0 : un argument nécessaire pour certaines instructions.

Voici quelques exemples d'instructions:

- "A4" : Stoppe la lecture du fichier son en cours de lecture.
- "A7 : 00000" : Lis le fichier cité dans la commande. Le terme "00000" doit être un nombre compris entre 00001 et 99999.
Par exemple, l'instruction "A7 : 12321" fait lire le fichier nommé "12321" s'il existe.
- "AF : 00" : Sélection du volume, compris entre 00, muet, et 30, maximum. Valeur par défaut : 20.
- "B0" : incrémente le volume si possible.
- "B1" : décrémente le volume si possible.

Le protocole I²C, pour Inter-integrated Circuit, ou circuit intégré, est un protocole de communication série synchrone maître-esclave développé par Philips depuis les années 1990. Ce protocole est de type half-duplex : la communication entre deux appareils peut se faire dans les deux sens, mais pas au même moment. Le protocole est basé sur deux signaux : un signal d'horloge SCL et un signal de données SDA.

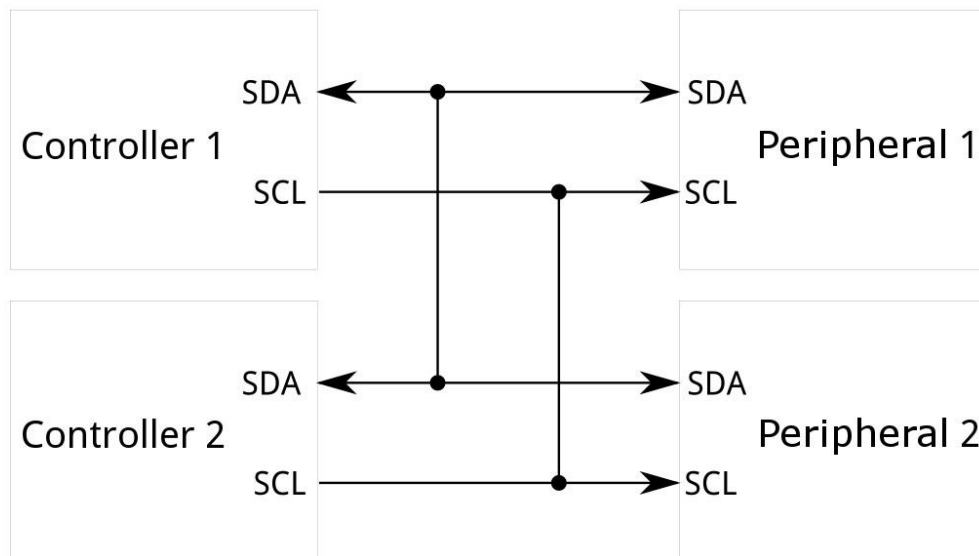


Fig. 5 : Schéma de liaison de maîtres et d'esclaves I²C.

Les maîtres génèrent le signal d'horloge tandis que les esclaves le reçoivent.

La trame est composée ainsi :

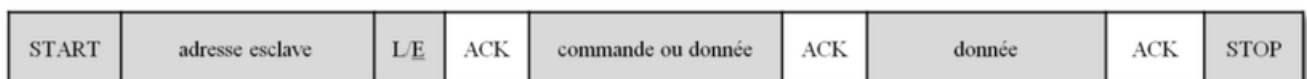


Fig. 6 : Trame d'une communication I²C.

- Signal de départ émis par le maître.
- Adresse de l'esclave concerné par le message, composé de 7 bits.
- Envoi d'un bit pour désigner si le maître va envoyer des données à l'esclave (bit à 0) ou s'il veut recevoir des données de la part de l'esclave (bit à 1).
- Acquiescement de la part de l'esclave. Si l'esclave a correctement reçu l'octet précédent, ce dernier enverra un bit 0 dans le signal de données, sinon, il laissera le bit à 1.
- Le maître enverra un octet pour informer dans un premier temps à l'esclave la commande à faire ou la donnée à renvoyer.
- L'esclave répond aux ordres du maître en lui renvoyant certaines informations si besoin ou envoie les données demandées par le maître.
- A chaque octet envoyé par le maître ou l'esclave, le receptr de l'octet doit envoyer un signal d'acquiescement pour confirmer la bonne réception du signal.
- A la fin de la trame, le maître envoie un signal de fin.

Le protocole SPI, pour Serial Peripheral Interface, ou interface périphérique série, est un protocole de communication série synchrone maître-esclave développé par Motorola depuis les années 1980. Ce protocole est de type full-duplex : la communication entre deux appareils peut se faire dans les deux sens et au même moment. Le protocole est basé sur au moins quatre signaux : un signal d'horloge SCLK, un signal pour envoyer les données du maître à l'esclave MOSI : Master Out Slave In, un signal pour envoyer les données de l'esclave au maître MISO : Master In Slave Out et un signal de sélection par esclave SS.

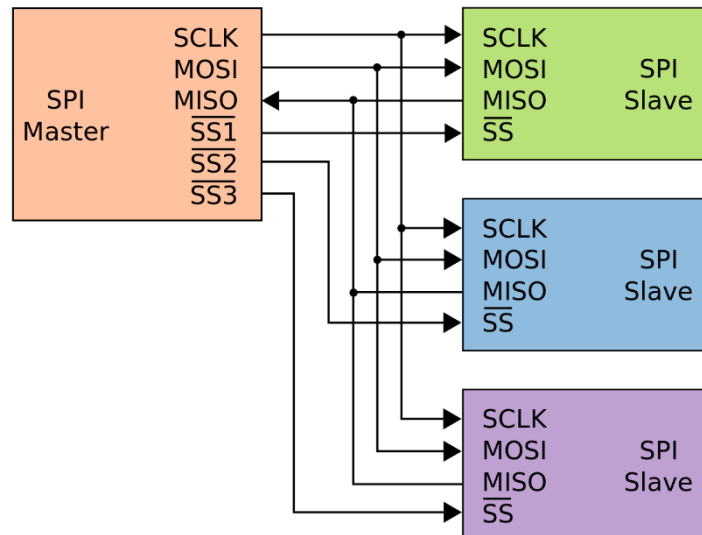


Fig. 7 : Schéma de liaison de maîtres et d'esclaves SPI.

Lorsque le maître veut communiquer avec un esclave, le signal SS de l'esclave en question est mis à l'état logique bas par le maître pour démarrer la communication. Ensuite, les deux appareils utilisent les signaux MOSI et MISO pour échanger des données. A chaque coup d'horloge, maître et esclave envoient un bit de données à l'autre.

Vous trouverez le code source de la première version en annexe 4.

Deuxième version

Lors des premières semaines de mon stage, j'ai découvert chaque composant de la maquette puis j'ai commencé à développer une deuxième version qui permettra de parler en morse avec les quatre faces du cube dotés d'un contact tactile. Pour écrire en morse, deux des contacts sont utilisés pour inscrire les caractères du morse : le point et le trait. Un troisième contact permet de passer au caractère suivant, en sachant que si le code morse est vide ou inconnu, un espace sera inscrit à la place. Enfin, le dernier contact permet de recommencer le code morse actuel, s'il a mal été inscrit par exemple. Si le code morse est vide, le dernier caractère sera supprimé et le contact a pour effet un retour arrière.

Au démarrage, L'écran essaie d'initialiser le MPR121. Si le capteur n'est pas reconnu au démarrage comme à l'utilisation, l'écran affiche ce message d'erreur :



Fig. 8 : Message d'erreur signifiant l'absence du MPR121.

Au démarrage, l'écran s'affiche ainsi :

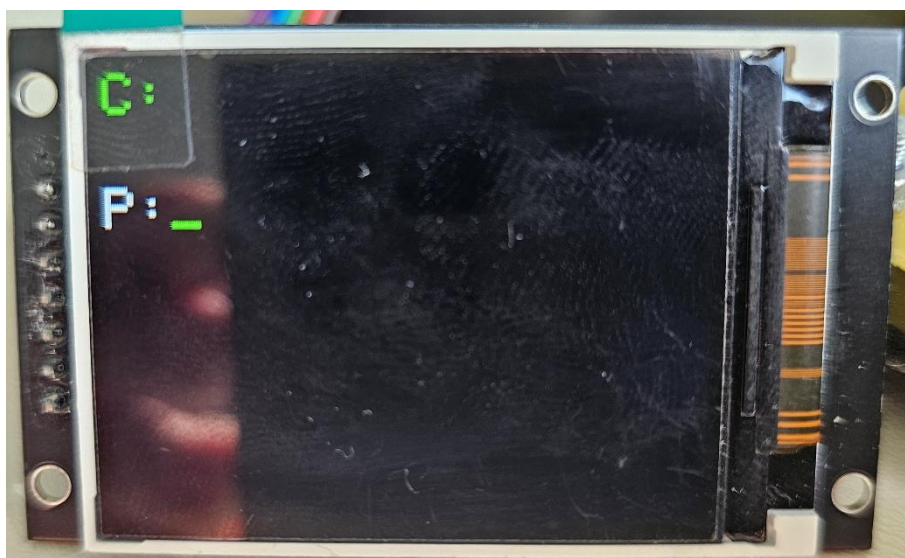


Fig. 9 : Ecran du système au démarrage.

La lettre P correspond à la phrase que l'utilisateur écrit et la lettre C correspond au code morse du caractère en cours d'écriture, représenté en vert sur la phrase.

En utilisant les contacts, on peut écrire une phrase comme montré ci-dessous :



Fig. 10 : Utilisation du système.

Si le code morse écrit pour le caractère ne correspond pas à un caractère existant dans le code morse, un point d'interrogation rouge sera affiché pour informer du code inconnu :

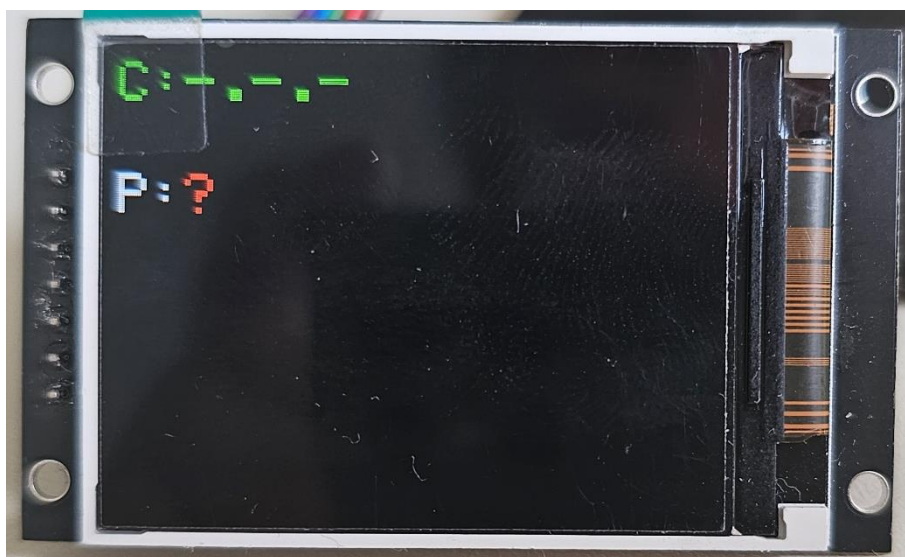


Fig. 11 : Réaction du système face à un code inconnu.

Si l'utilisateur appuie sur la touche pour passer au caractère suivant, le caractère sera supprimé comme s'il avait appuyé sur retour arrière pour recommencer ce caractère.

La mise en place de cette deuxième version n'était pas une source de difficulté pour moi. La découverte de chaque composant permet de bien les connaître et de savoir comment les programmer.

Vous trouverez le code source de la deuxième version en annexe 5.

Programmation par PlatformIO

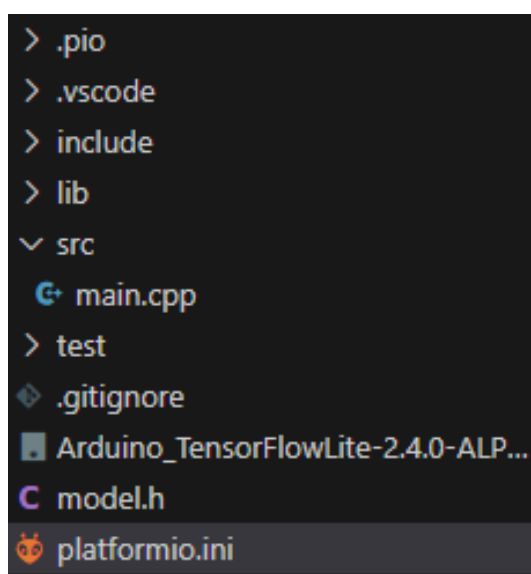
J'ai développé cette version en utilisant non pas le logiciel officiel de Arduino mais une extension sur le logiciel Visual Studio Code, nommé PlatformIO.

PlatformIO dispose de plusieurs fonctionnalités qui facilitent la programmation d'une carte Arduino. Parmi ces fonctionnalités, on trouve la possibilité de programmer en langage C++. Normalement ces cartes ne peuvent se programmer d'en langage C mais PlatformIO nous permet de programmer en C++. Un autre exemple de fonctionnalité est un fichier nommé platformio.ini dans lequel se trouve les informations les plus importantes telles que la carte utilisée et les librairies utilisées et leurs versions, ce qui permet en cas de changement d'ordinateur par exemple, de retélécharger automatiquement les librairies utilisées dans les versions voulues, même s'il ne s'agit plus de la version la plus récente.

```
[env:nano33ble]
platform = nordicnrf52
board = nano33ble
framework = arduino
lib_deps =
  C:\Users\Etudiants\Documents\PlatformIO\Projects\Test\Arduino_TensorFlowLite-2.4.0-ALPHA.zip
  arduino-libraries/Arduino_BMI270_BMM150@^1.2.0
  arduino-libraries/ArduinoBLE@^1.3.6
  adafruit/Adafruit_MPR121@^1.1.3
```

Fig. 12 : Contenu d'un fichier platformio.ini.

Comme montré ci-dessus, les librairies utilisables peuvent être soit une librairie téléchargeable dans PlatformIO, soit une librairie externe à PlatformIO,



```
> .pio
> .vscode
> include
> lib
v src
  main.cpp
> test
  .gitignore
  Arduino_TensorFlowLite-2.4.0-ALP...
  model.h
  platformio.ini
```

Fig. 13 : Contenu d'un dossier de projet sur PlatformIO.

Comme pour le logiciel officiel Arduino, les projets sont répartis dans leur dossier. Ici, chaque dossier contient divers éléments. Voici le contenu de certains d'entre eux.

Le sous-dossier « .pio » stocke des fichiers importants pour la compilation tels que les fichiers des librairies.

Le sous-dossier « src » contient tous les fichiers de programmations. Il est possible d'en faire plusieurs pour mieux se repérer : un par fonctionnalité importante et un principal faisant le lien par exemple.

Enfin, le fichier platformio.ini, ainsi que le fichier .zip d'une librairie et un fichier d'en-tête sont également présents, mais non attribués à un sous-dossier.

Troisième version

Pendant que la deuxième version est en développement, la première a été testée par Aurélie. Malheureusement, les tests n'ont pu aboutir à cause de l'état de santé d'Aurélié qui ne lui a pas permis de beaucoup essayer la première version.

Nous avons donc continué le développement du projet sans les résultats du test. Nous avons préparé la troisième version par une amélioration ergonomique : le cube devient une sphère, une forme plus agréable à tenir et qui permet de placer cinq contacts, une par doigt. Cette forme a été modélisée en trois dimensions par Sébastien Carrière et j'ai pu observer avec Pascal Hussaud le processus de modélisation, chose que je n'ai jamais fait. Cette matinée m'a permis de découvrir comment modéliser un objet et toutes les spécificités qui en découlent comme comment prévoir la place pour chaque élément prévu dans la sphère. La sphère a été ensuite imprimée par Christophe Malinowski.



Fig. 14 : Vue externe de la demi-sphère supérieure avec les contacts.

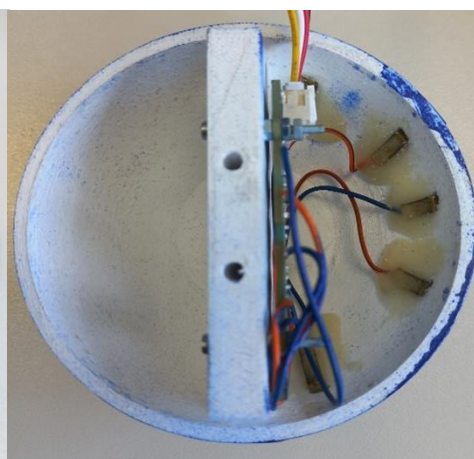


Fig. 15 : Vue interne de la demi-sphère supérieure avec le MPR121 intégré.

De plus, la carte Arduino est un modèle plus performant : un Arduino Nano 33 BLE Sense Rev2. Cette carte contient un microprocesseur Nordic nRF52840, plus performante que la ATmega328 des Arduino Nano classiques. L'Arduino fonctionne désormais sur batterie via circuit imprimé permettant également d'y connecter le MPR121 ainsi que l'écran ST7789 pour effectuer du débogage. Un bouton poussoir est également présent dans la même utilité.



Fig. 16 : Carte sur laquelle repose l'Arduino avec les connecteurs et le

Grâce aux capteurs internes de l'Arduino dont font parties un accéléromètre et un gyroscope, ainsi qu'aux performances du Nordic nRF52840, une reconnaissance de mouvements via intelligence artificielle pourra être ajoutée. La mise en place de l'intelligence artificielle est une chose que je n'ai jamais faite. Enfin, pour rendre la sphère sans fil, il faut afficher ce que l'utilisateur écrit de manière sans fil. Pour cela, une communication Bluetooth avec un smartphone sera à mettre en place, une chose que je n'ai jamais faite non plus. Il me fallait alors découvrir puis développer ces deux fonctionnalités.

Reconnaissance des mouvements

Pour détecter les mouvements et interagir en fonction, je vais utiliser les capteurs internes de l'Arduino et de l'apprentissage automatique. Je vais utiliser le service Google TensorFlow pour cet apprentissage.

L'Arduino utilise un IMU, pour Inertial Measurement Unit, aussi appelé centrale à inertie, qui est un ensemble de trois accéléromètres, de trois gyroscopes et dans certains cas, dont celui de l'Arduino, de trois magnétomètres, capable d'effectuer des mesures basées sur les positions et les vitesses comme l'accélération ou les angles. Les capteurs sont par trinômes pour mesurer dans les trois dimensions.

Pour lire les mesures des capteurs, j'ai utilisé une

L'apprentissage automatique, plus souvent nommé par son terme anglais « machine learning », est un outil basé sur l'intelligence artificielle, qui permet de discerner des éléments spécifiques dans les données pour différencier plusieurs éléments entre eux. Dans le projet, je vais lui donner les données fournies par l'accéléromètre et le gyroscope de l'Arduino pour qu'il puisse reconnaître les mouvements effectués par l'utilisateur.

L'implémentation du système TensorFlow nécessite diverses étapes. La première consiste à récupérer les informations de l'accéléromètre et du gyroscope sur des échantillons constitués de 119 mesures des deux capteurs sur une période. Pour améliorer la capacité de reconnaître les mouvements, il faut effectuer les mouvements plusieurs fois, très légèrement différentes, en faisant un échantillon à chaque fois.

Ensuite, tous les échantillons de chaque mouvement sont réunis sur un même fichier csv. Tous les fichiers csv seront ensuite données à TensorFlow.

Un fichier csv, pour Comma-Separated Values, est un fichier contenant des valeurs sous forme d'un tableau, chaque valeur étant séparée par une virgule. Ici, les fichiers contiennent en première ligne la signification des colonnes puis les échantillons, chaque échantillon étant séparée par une ligne vide.

```
aX,aY,aZ,gX,gY,gZ
-0.903,-0.004,-0.495,-19.165,-90.942,-7.324
-0.891,-0.028,-0.515,-10.925,-84.717,-0.671
-0.877,-0.065,-0.522,-10.376,-82.947,-0.061
-0.869,-0.075,-0.536,-13.000,-86.792,-4.883
-0.839,-0.072,-0.558,-12.695,-90.637,-9.644
-0.823,-0.064,-0.566,-8.606,-93.079,-17.395
-0.780,-0.051,-0.592,-6.226,-92.041,-19.836
-0.753,-0.058,-0.611,-2.991,-83.923,-22.095
-0.716,-0.071,-0.623,-3.906,-79.224,-23.254
-0.703,-0.072,-0.621,-13.000,-79.468,-24.780
-0.700,-0.024,-0.630,-19.775,-85.083,-22.705
-0.705,0.001,-0.653,-29.236,-98.145,-10.498
-0.705,-0.004,-0.724,-28.625,-99.365,-5.737
-0.709,-0.017,-0.746,-20.447,-90.027,-4.578
-0.717,-0.032,-0.753,-18.066,-82.703,-6.042
-0.703,-0.035,-0.760,-21.667,-66.956,-5.066
-0.656,-0.040,-0.782,-25.696,-57.495,-0.488
-0.635,-0.049,-0.780,-31.372,-41.260,11.169
-0.628,-0.052,-0.780,-32.959,-39.612,14.526
-0.632,-0.044,-0.805,-30.090,-44.006,15.625
-0.647,-0.052,-0.835,-23.560,-42.725,13.855
-0.655,-0.066,-0.816,-9.949,-33.081,8.362
-0.642,-0.077,-0.795,-8.484,-29.907,8.301
-0.620,-0.084,-0.811,-10.315,-24.902,13.245
-0.601,-0.078,-0.844,-9.216,-22.034,14.343
-0.607,-0.089,-0.841,-3.845,-17.578,13.489
-0.618,-0.104,-0.849,-3.296,-16.724,12.573
-0.635,-0.081,-0.869,-3.784,-10.437,13.367
-0.655,-0.035,-0.907,0.671,-0.488,15.564
```

Fig. 17 : Début du fichier csv contenant la description des colonnes et les premières mesures du premier échantillon.

Via plusieurs programmes Python, TensorFlow est capable de lire les fichiers csv, de discerner les gestes spécifiques effectuées lors des mouvements et de créer un fichier d'en-tête qui sera utilisé sur le programme final. Ce fichier permet à l'Arduino d'utiliser les apprentissages de l'intelligence artificielle afin de reconnaître les mouvements et d'interagir en fonction. D'autres programmes permettent d'obtenir diverses informations sur l'apprentissage comme la capacité d'apprentissage de l'intelligence artificielle.

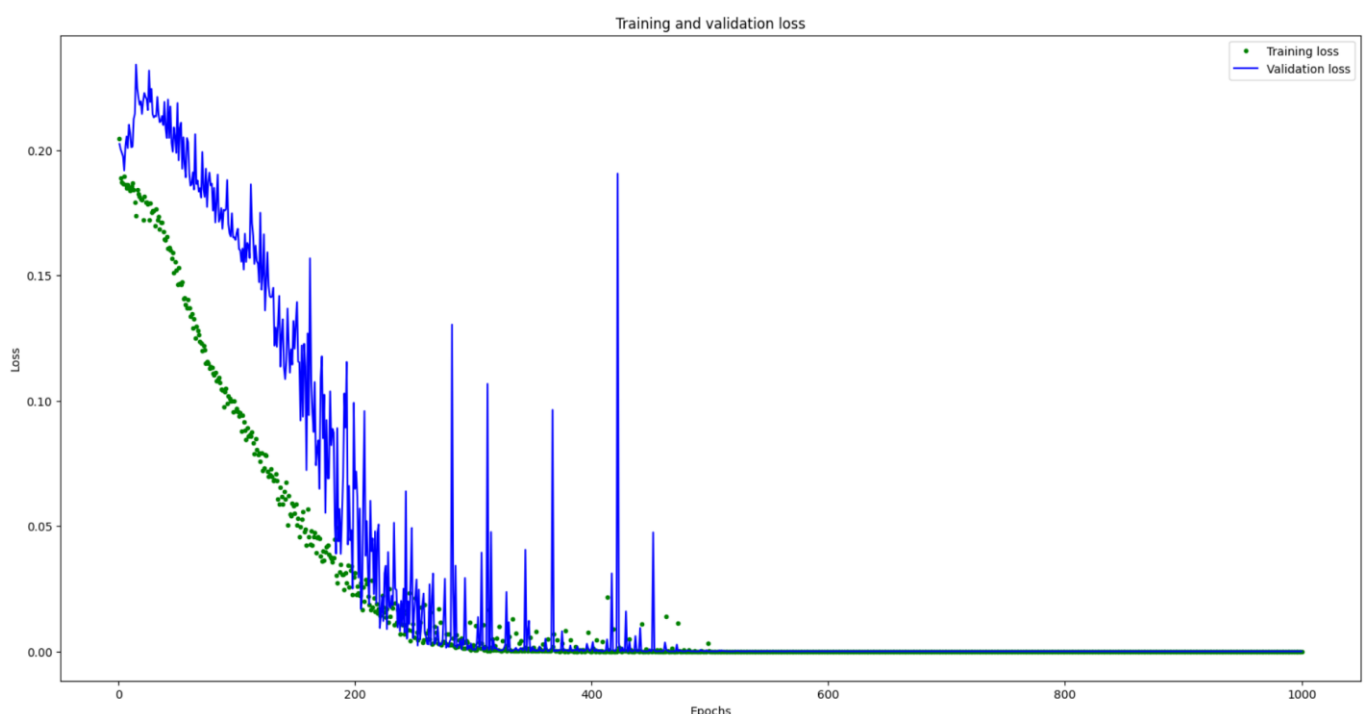


Fig. 18 : Taux d'erreur de l'IA au fur et à mesure des étapes de recherche. Plus cette valeur est petite, plus l'IA devient performante.

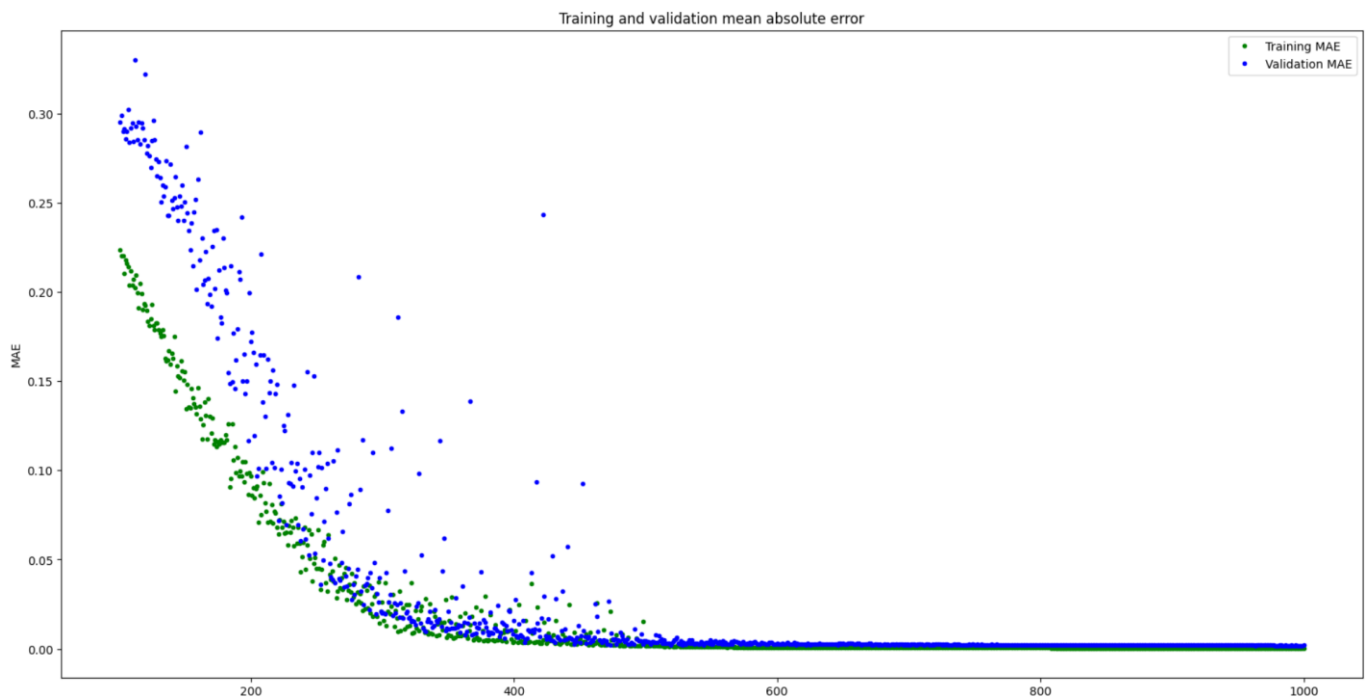


Fig. 19 : Erreur moyenne absolue de l'IA au fur et à mesure des étapes de recherche. Plus cette valeur est petite, plus l'IA devient performante.

Une fois le fichier d'en tête intégré au programme final, l'Arduino est désormais capable de reconnaître les mouvements qu'on lui aura donné via les fichiers csv.

Pour effectuer ces mouvements, l'utilisateur devra simplement commencer naturellement un des mouvements. La somme des accélérations sera alors supérieure à une certaine valeur précisée dans le programme, qui fera commencer la mesure du mouvement.

Dans un premier temps, j'ai mis en place la reconnaissance parmi deux mouvements : un coup de poing et une flexion du coude. Ceci avait pour but de vérifier le bon fonctionnement de l'intelligence artificielle et de la découvrir à l'aide d'un exemple. Celui-ci étant concluant, je suis passé à la reconnaissance parmi quatre mouvements : lever le poignet, baisser le poignet, tourner la main et orienter la main vers la droite. Ces mouvements sont respectivement appelés lever, baisser, tournis et droite.

Ci-contre, le terminal série indique qu'il a détecté quatre mouvements. Pour chacune d'entre elles, l'Arduino effectue un échantillonnage comme pour l'acquisition des données puis utilise le fichier d'en-tête pour déterminer quel mouvement a été effectué. Il calcule la probabilité que chaque mouvement enregistré correspond à celui effectué et considère le plus probable comme le mouvement réalisé.

Chaque mouvement effectuera une action, qui peut également être réalisé par le capteur MPR121. L'utilisateur pourra alors choisir d'effectuer ces actions ou touchant ou en bougeant la sphère. L'utilisateur peut aussi combiner les deux méthodes.

```
Début mesure
baisser: 0.000000
droite: 0.353647
lever: 0.645438
tournis: 0.000915
Je pense que c'est : lever avec une valeur de 0.65

Début mesure
baisser: 0.991034
droite: 0.008966
lever: 0.000000
tournis: 0.000000
Je pense que c'est : baisser avec une valeur de 0.99

Début mesure
baisser: 0.000001
droite: 0.922710
lever: 0.007271
tournis: 0.070018
Je pense que c'est : droite avec une valeur de 0.92

Début mesure
baisser: 0.000000
droite: 0.229001
lever: 0.001506
tournis: 0.769493
Je pense que c'est : tournis avec une valeur de 0.77
```

Fig. 20 : Terminal série de l'Arduino, montrant les mouvements détectés.

Pour augmenter la précision de la détection, il est possible de mettre en place diverses choses. Par exemple, il est possible d'ajouter la détection d'un mouvement léger qui n'a aucun effet. Dans la même idée, on peut ne pas effectuer d'interaction si le mouvement lié à une probabilité trop faible : dans le cas de l'image ci-dessus, le premier mouvement détecté à une probabilité de détection de 65%, ce qui reste relativement faible. Dans ce cas, nous pourrions ne pas faire réagir le mouvement car il est trop peu probable même s'il est le plus haut parmi les quatre mouvements.

Nouveau moyen de communication

Pour permettre à plus de personne de communiquer à l'aide de la sphère, nous avons décidé de mettre en oeuvre un autre moyen de communication que le morse pour permettre à plus d'utilisateurs de l'utiliser. Désormais, nous utiliserons une communication semblable aux téléphones à touches.

Sur les cinq contacts, ceux de l'index, du majeur et de l'annulaire sont utilisés pour choisir la lettre. Pour choisir la lettre, il suffit de toucher une ou plusieurs fois un de ces contacts. Puis, pour passer à la lettre suivante, il suffit de toucher le contact du pouce. Si ce contact est touché sans avoir sélectionné de lettre, un espace sera inséré à la place. Enfin, le contact de l'auriculaire est utilisé comme touche de retour arrière.

Ci-dessous, l'ordre de sélection des lettres :

Contacts consécutifs	1	2	3	4	5	6	7	8	9
Index	a	e	i	o	u	y	w	x	z
Majeur	b	c	d	f	g	h	j	k	
Annulaire	l	m	n	p	q	r	s	t	v

Fig. 21 : Tableau des caractères du nouveau moyen de communication.

Par exemple, si j'appuie 7 fois de suite sur le capteur du majeur, la lettre sélectionnée sera « j ». Si j'appuie sur un autre contact, le compteur sera remis à 1. Donc dans l'exemple, si j'appuie ensuite sur le capteur de l'index, la lettre sélectionnée sera maintenant « a ». Il me suffira enfin d'appuyer sur le contact du pouce pour insérer un « a » dans la phrase que j'écris.

Un axe d'amélioration de ce système est de prendre en compte les accents et la cédille, ce qui demanderait une limite d'appuis plus longue ou de pouvoir sélectionner la lettre avec quatre contacts, en modifiant l'effet du contact du pouce d'un passage au caractère suivant à un quatrième contact de sélection de caractère.

Communication Bluetooth

Pour permettre de rendre la sphère réellement sans fil, il est nécessaire de mettre en place une communication sans fil pour afficher ce que l'utilisateur écrit. Il existe plusieurs moyens de communication et appareils récepteur qui pourraient afficher ce qui est écrit. Nous avons choisi une communication par Bluetooth Low Energy et sur un smartphone. Nous utiliserons le smartphone car il s'agit d'un appareil que tout le monde possède et qu'il nous suffit d'utiliser une application existante, nous aurons simplement besoin d'ajouter au programme de la sphère la communication sans la nécessité de développer un second appareil ni un second programme pour ajouter la communication.

L'implémentation du Bluetooth a été une étape difficile car il s'agit d'un moyen de communication que je n'ai jamais manipulé ni étudié en profondeur. De plus, trouver une application du type terminal Bluetooth compatible et fonctionnelle a été compliquée. Après plus d'une dizaine d'applications essayées, la première qui a fonctionné totalement m'a été montrée par une exposante du douzième salon handicap et technologies, dont je vous fais la présentation dans le chapitre suivant et dont nous avons exposé notre projet.

Après avoir trouvé une application fonctionnelle, j'ai remarqué que le programme que j'avais réussi à mettre en place ne pouvait envoyer qu'un caractère à la fois. Pour corriger ce problème et envoyer une chaîne de caractère, j'ai dû changer la quasi-totalité du programme concernant le Bluetooth.

Le terminal affichera la lettre sélectionnée lorsqu'il appuie sur un contact pour en choisir une. Lorsque le contact du caractère suivant ou de retour arrière est touché, la phrase est affichée. Enfin, lors du départ de la mesure afin de détecter le mouvement, le mot « Mesure » sera affiché, puis à la fin de la reconnaissance, le mouvement détecté est affiché avec sa probabilité de détection.

```
DEBUG: Connected to GATT server
Mesure
Lever (93.85%)
Mesure
Baisser (99.59%)
Mesure
Tournis (76.53%)
Mesure
Droite (99.94%)
```

Fig. 22 : Réaction des mouvements de la sphère vue sur le terminal Bluetooth.

A noter qu'ici, les deux méthodes d'interactions sont montrées séparément mais peuvent être utilisées ensemble.

```
DEBUG: Connected to GATT server
lettre : b
lettre : c
lettre : d
lettre : f
lettre : g
lettre : h
Phrase : h.
lettre : a
lettre : e
Phrase : he.
lettre : l
Phrase : hel.
lettre : l
Phrase : hell.
lettre : a
lettre : e
lettre : i
lettre : o
Phrase : hello.
Phrase : hell.
Phrase : hel.
Phrase : he.
Phrase : h.
Phrase : .
```

Fig. 23 : Réaction des contacts de la sphère vue sur le terminal Bluetooth.

Vous trouverez le code source de la troisième version en annexe 6.

Présentation au projet au douzième Challenge Handicap et Technologie

Nous avons eu l'occasion de présenter le projet lors du douzième challenge handicap et technologie à l'Ecole Nationale d'Ingénieurs de Metz les 23 et 24 mai. Ce challenge m'a permis de montrer le projet et ce que nous avons réalisé à un public composé de diverses personnes : en dehors du jury, le public comprenait des enseignants, des techniciens et des étudiants de l'école accueillante et des autres écoles de la zone ainsi que des lycéens, dont certains étaient aussi candidats.

La présentation du projet a été un exercice bénéfique pour moi pour améliorer ma capacité à parler à un public. J'ai pu également discuter avec d'autres exposants notamment une chercheuse de l'université Paris 8, qui a pu m'apporter de l'aide pour la communication Bluetooth (cf. Annexe 1).

Le projet a été apprécié par le jury et a obtenu le deuxième titre dans la catégorie Communication et Accessibilité numérique.

Parmi les autres projets présentés au challenge, il y avait :

- Un boîtier à placer entre sa tête et une casquette, sur le bras via un élastique ou sous une plateforme sur laquelle l'utilisateur se tient dessus en équilibre pour rééduquer par le jeu respectivement le cou, les mains ou l'équilibre.
- Des modules permettant à diverses personnes de pouvoir utiliser un smartphone, un ordinateur ou une console de jeu. Ces modules sont sous diverses formes comme un capteur de force ou un joystick à placer sur le fauteuil roulant.
- Une imprimante permettant d'imprimer un texte en braille sur une feuille de papier.

Conclusion

Ce projet est d'une grande aide à Aurélie, qui lui était uniquement prédestiné. Nous avons décidé durant le développement du projet d'en agrandir les utilisateurs potentiels en modifiant la manière de former les phrases.

Le système est relativement large et pourrait être destiné à beaucoup de public en adaptant à chacun : la position des contacts pourrait être adaptés selon la morphologie des mains des utilisateurs et pourraient donc être utilisés pour des personnes qui ne peuvent bouger uniquement leur main droite.

Le moyen d'écrire peut aussi être adapté aux capacités de l'utilisateur : nous avons commencé par une écriture basée sur du code morse puis nous l'avons changé par un système par sélection de lettre.

La langue peut être adaptée aussi en changeant les lettres accessibles. Par exemple, en ajoutant la possibilité d'écrire la lettre ñ, l'utilisateur pourrait écrire en français et en espagnol.

Deux axes d'amélioration matériel seraient d'ajouter un bouton d'alimentation sur la sphère ainsi qu'un système de rechargement de la batterie sans nécessité de la sortir de la sphère.

Ce stage m'a permis de découvrir diverses nouvelles notions dans la programmation d'Arduino comme l'intégration de la communication Bluetooth ou encore de l'intelligence artificielle. De plus, le douzième challenge handicap et technologie a été une source de rencontres et de communications avec des personnes du domaine.

Pour terminer, je tiens à remercier :

- Larbi Chrifi pour m'avoir permis de réaliser mon stage au Laboratoire des Technologies Innovantes,
- Sébastien Carrière pour avoir été mon tuteur de stage et pour avoir effectué les modélisations en 3D pour le projet,
- Pascal Hussaud pour le développement des cartes et des maquettes du projet,
- Christophe Malinowski pour le montage des cartes du projet,
- Abdel Faqir pour le support informatique et l'aide apportée,
- Pascale Brocard pour les tâches administratives envers le stage.

Annexe 1

RE : Avancée projet "le Cube"



theo.dhuiege@gmail.com
celine.jost@gmail.com

29 mai



Bonjour,

Je me nomme Théo Dhuiege, j'ai exposé mon projet « le Cube » lors du 12^e challenge handicap et technologie.

Je reviens vers vous pour vous donner des nouvelles et vous demander des questionnements.

J'ai pu réussir à mettre en place une communication Bluetooth vers le smartphone avec l'application que vous m'avez conseillé et pour cela je vous en remercie.

Nous sommes encore en recherche pour savoir quelle direction prendre pour notre projet : continuer sur de la communication ou changer pour du contrôle d'appareils, de la rééducation ou autre.

Auriez-vous des conseils ou des connaissances qui pourraient nous aider dans notre réflexion et dans la suite du projet ?

Je mettrai à jour mon CV fin juin pour y inclure le projet, souhaiteriez-vous le recevoir ?

Je vous remercie de votre attention et de vos réponses.

En vous souhaitant une bonne journée.

Théo Dhuiege, BUT GEII 2^e année, Soissons-Cuffies.



Céline JOST
theo.dhuiege@gmail.com

29 mai



Bonjour Théo,

Ah je suis contente de voir que la communication Bluetooth est réglée. C'est une excellente nouvelle.

Pour pouvoir répondre à ta deuxième question, il me faudrait idéalement la photo du cube et une petite explication pour que je puisse transmettre à quelques collègues et voir qui réagit.

Et pour le CV oui je veux bien le recevoir. Je pourrais ainsi le distribuer à mes collègues :)

Je te souhaite également une très bonne suite de journée.

Céline.

Fig. 24 : Communication avec la chercheuse de l'université Paris 8 rencontré au challenge.

Annexe 2

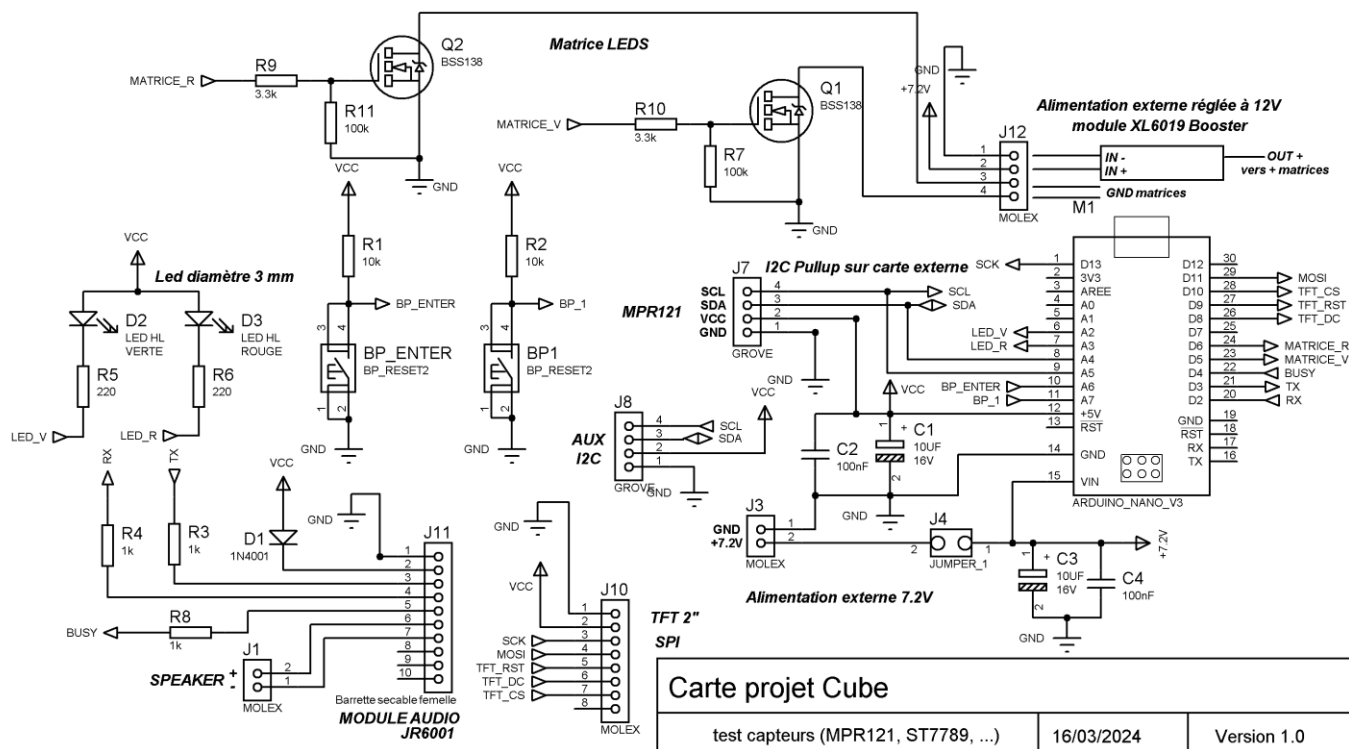


Fig. 25 : Schéma électronique de la première et de la deuxième version.

Annexe 3

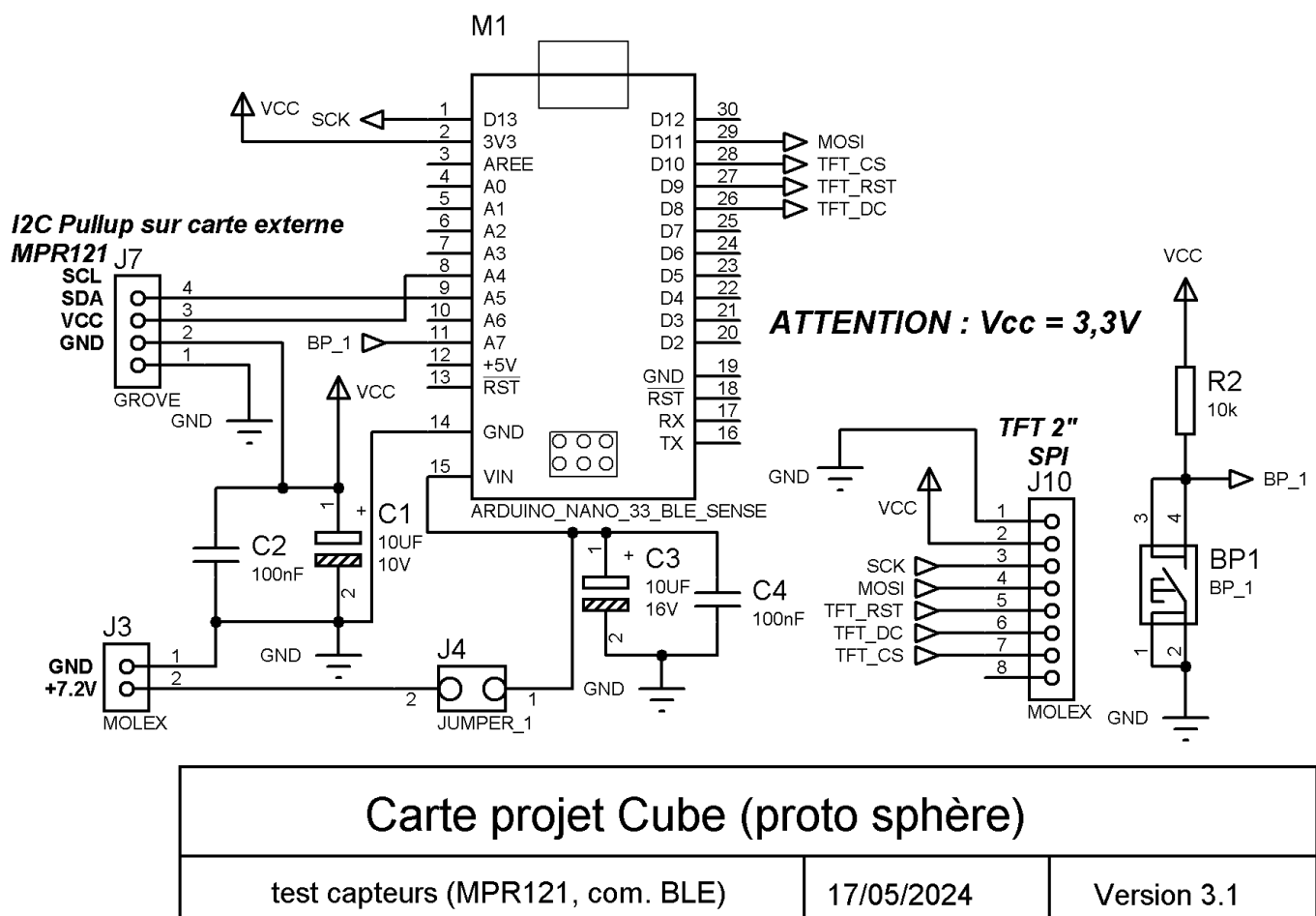


Fig. 26 : Schéma électronique de la troisième version.

Annexe 4

```
/* *****
* CUBE_V1.c 18/03/2024 *
* *****
* Debug sur UART (115200,8,N,1) *
* Capteur MPR121 connecté sur 4 faces du cube (récupération d'un n°0 à 3 *
* Capteur MGC3130 en tablette avec récupération des 3 positions x,y et z *
* Ecran TFT 2" utilisé pour afficher un point ou un trait *
* LED RGB WS2812 non utilisée *
* *****
* * BP_ENTER et BP1 sont connectés sur A6 et A7 (entrée uniquement analog *
* *
*/

/* GroveI2CTouchTest.pde - Sample code for the SeeedStudio Grove I2C Touch Sensor
http://seeedstudio.com/wiki/index.php?title=Twig_-_I2C_Touch_Sensor_v0.93b
Prerequisite: A modification of the Grove I2C Touch sensor is needed Solder a pin to
the INT terminal in the I2C sensor to connect it to one pin in Arduino Created by
Wendell A. Capili, August 27, 2011. http://wendellinfinity.wordpress.com
Released into the public domain.*/

#include <Wire.h> // include I2C library
#include <i2c_touch_sensor.h> // include our Grove I2C touch sensor library
#include <MPR121.h> // include MPR121 library
// IMPORTANT: in this case, INT pin was connected to pin7 of
// the Arduino
// (this is the interrupt pin)
#define MatriceR 6 // définition matrice Rouge broche D6 (active High)
#define MatriceV 5 // définition matrice Verte broche D5 (active High)
#define LedR A3 // définition matrice Rouge broche A3 (active Low)
#define LedV A2 // définition matrice Verte broche A2 (active Low)
#define BP_1 A7 // A7 et A6 sont des entrées exclusivement analogiques
#define BP_ENTER A6

i2ctouchsensor touchsensor; // déclaration de l'objet
boolean padTouched[4]; // Tableau contenant l'état des 4 touches du cube
long previousMillis = 0; // variable pour gestion du temps avec l'échantillonnage
long interval = 500; // définition de la durée d'échantillonnage (500ms)
/*****
* Setup *
*****/

void setup()
{
    int i; // Variable de boucle

    pinMode(16, OUTPUT); // A2 : LED_V
    digitalWrite(LedV, HIGH); // Extinction LED Verte
    pinMode(17, OUTPUT); // A3 : LED_R
    digitalWrite(LedR, HIGH); // Extinction LED Rouge
    pinMode(7, OUTPUT); // D7 : Signal WS2812
    pinMode(6, OUTPUT); // D6 : commande matrice ROUGE (active à 0L)
    digitalWrite(MatriceR, LOW); // Extinction matrice Rouge
    pinMode(5, OUTPUT); // D5 : commande matrice VERTE (active à 0L)
    digitalWrite(MatriceV, LOW); // Extinction matrice Verte
    pinMode(4, INPUT); // D4 : BUSY du JR6001
    Serial.begin(115200); // Config UART
    Serial.print("Debut");
    Wire.begin(); // Needed by the GroveMultiTouch lib
    touchsensor.initialize(); // Initialize the Grove I2C touch sensor
    for(i=0; i<=3; i++) // Initialize tableau d'état des touches
        padTouched[i]=false;
    digitalWrite(MatriceR, HIGH); // Activation matrice Rouge
    delay(1500);
    digitalWrite(MatriceR, LOW); // Extinction matrice Rouge
    digitalWrite(MatriceV, HIGH); // Activation matrice Verte
    delay(1500);
    digitalWrite(MatriceV, LOW); // Extinction matrice Verte
    // buttonState = digitalRead(buttonPin);
}
/*****
* Lecture_BP *
*****/

char Lecture_BP ()
{
    unsigned int val_BP_1, val_BP_ENTER;
    char valeur=0;

    val_BP_1 = analogRead(BP_1);
    Serial.print(val_BP_1);
    val_BP_ENTER = analogRead(BP_ENTER);
}
```

```

    Serial.print(val_BP_ENTER);
    if(val_BP_1>4000) bitSet(valeur,0);
        else if(val_BP_1<100) bitClear(valeur,0);
    if(val_BP_ENTER>4000) bitSet(valeur,1);
        else if(val_BP_1<100) bitClear(valeur,1);
    return (valeur);
}
void loop()
{
    unsigned char MPR_Query=0;
    unsigned long currentMillis = millis();           // valeur courante de la variable interne lue
    if(currentMillis - previousMillis > interval)     //SI le temps écoulé est supérieur à l'intervale
                                                        défini (100ms)
    {
        previousMillis = currentMillis;             //ALORS on affecte la valeur ancienne à la valeur actuelle
    //    touchsensor.getTouchState();
        touchsensor.readTouchInputs();
    }
    for (int i=0;i<12;i++)
    {
        if (touchsensor.touched&(1<<i))
        {
            Serial.print("pin ");
            Serial.print(i);
            Serial.println(" was touched");
        }
    }
}
}

```

Annexe 5

```
/* *****
 * CUBE_V2.c                avril-mai 2024                *
 * *****
 * Capteur MPR121 connecté sur 4 faces du cube (récupération d'un n°0 à 3) *
 * Ecriture et décodage du morse                *
 * Ecran TFT 2" utilisé pour afficher un point ou un trait                *
 * *****/
#include <Arduino.h> //Mise en place des librairies
#include "Adafruit_MPR121.h"
#include <SoftwareSerial.h>
#include <Adafruit_ST7789.h>
#include <DFRobot_MGC3130.h>
#include <SPI.h>
#include <string.h>
using namespace std;

#define TFT_CS 10
#define TFT_RST 9 // Or set to -1 and connect to Arduino RESET pin
#define TFT_DC 8
Adafruit_ST7789 tft = Adafruit_ST7789(TFT_CS, TFT_DC, TFT_RST);

#define MatriceR 6 // définition matrice Rouge broche D6 (active High)
#define MatriceV 5 // définition matrice Verte broche D5 (active High)
#define LedR A3 // définition matrice Rouge broche A3 (active Low)
#define LedV A2 // définition matrice Verte broche A2 (active Low)
#define BP_1 A7 // A7 et A6 sont des entrées exclusivement analogiques
#define BP_ENTER A6
#define Busy 4

DFRobot_MGC3130 myGesture(14, 15); // DPin : 8 ; MCLRPin : 9
// You can have up to 4 on one i2c bus but one is enough for testing!
Adafruit_MPR121 cap = Adafruit_MPR121();
// Liaison série émulée pour dialogue JR6001
SoftwareSerial Serial_JR6001(2, 3); // RX : D2 ; TX : D3

// color definitions
const uint16_t Display_Color_Black = 0x0000;
const uint16_t Display_Color_Blue = 0x001F;
const uint16_t Display_Color_Red = 0xF800;
const uint16_t Display_Color_Green = 0x07E0;
const uint16_t Display_Color_Cyan = 0x07FF;
const uint16_t Display_Color_Magenta = 0xF81F;
const uint16_t Display_Color_Yellow = 0xFFE0;
const uint16_t Display_Color_White = 0xFFFF;

// The colors we actually want to use
uint16_t Display_Text_Color = Display_Color_Yellow;
uint16_t Display_Background_Color = Display_Color_Black;

// assume the display is off until configured in setup()
bool isDisplayVisible = true;

// declare size of working string buffers. Basic strlen('d hh:mm:ss') = 10
const size_t MaxString = 16;

uint16_t touched = 0; //La mesure du MPR121 sera stockée ici
String i = "", phr = "", mot = ""; //Code morse et phrase écrite
char j = 0; //caractère reconnu
bool error = false;

void Morse(), MPR();

void MPR() { //Lecture MPR
    touched = cap.touched();
    if (touched == 4095) { //Si le capteur est déconnecté ou si les 12 contacts sont touchés
        //Ce qui est normalement impossible ici
        tft.fillScreen(Display_Color_Black);
        tft.setTextColor(Display_Color_Red);
        tft.setTextSize(4);
        tft.setCursor(0,0);
        tft.print("Erreur : \n capteur de \n contact \n deconnecte");
        while (touched == 4095) {
            touched = cap.touched();
            delay(50);
        }
        tft.setTextSize(3);
        tft.fillScreen(Display_Color_Black);
    }
}
```

```

    cap.begin(0x5B);
    Morse();
}
}

void Morse() //reconnaissance du caractère écrit
{
    error = false;
    j = ' ';
    if (i == "") //associe un caractère selon le code morse
        j = ' ';
    if (i == ".-")
        j = 'a';
    if (i == "-...")
        j = 'b';
    if (i == "-.-.")
        j = 'c';
    if (i == "-..")
        j = 'd';
    if (i == ".")
        j = 'e';
    if (i == "...")
        j = 'f';
    if (i == "--.")
        j = 'g';
    if (i == "....")
        j = 'h';
    if (i == "..")
        j = 'i';
    if (i == ".---")
        j = 'j';
    if (i == "-.-")
        j = 'k';
    if (i == "-...")
        j = 'l';
    if (i == "---")
        j = 'm';
    if (i == "-.-")
        j = 'n';
    if (i == "----")
        j = 'o';
    if (i == "...")
        j = 'p';
    if (i == "--.-")
        j = 'q';
    if (i == "-.-.")
        j = 'r';
    if (i == "...")
        j = 's';
    if (i == "-")
        j = 't';
    if (i == "-..")
        j = 'u';
    if (i == "....")
        j = 'v';
    if (i == "...")
        j = 'w';
    if (i == "-...")
        j = 'x';
    if (i == "-.-")
        j = 'y';
    if (i == "--..")
        j = 'z';
    if (i == "----")
        j = '1';
    if (i == "...")
        j = '2';
    if (i == "....")
        j = '3';
    if (i == "....")
        j = '4';
    if (i == "....")
        j = '5';
    if (i == "-....")
        j = '6';
    if (i == "--...")
        j = '7';
    if (i == "----.")
        j = '8';
    if (i == "----.")

```

```

j = '9';
if (i == "-----")
j = '0';
if (i == "...-.-.")
j = '?';
if (i == "-.-.-.-")
j = '!';
if (i == ".-.-.-.-")
j = '.';
if (i == "---.-.-")
j = ',';
if (i == "-.-.-.-.")
j = ';';
if (i == "-----.")
j = ':';
if (i == ".-.-.-.")
j = '+';
if (i == "-.....-")
j = '-';
if (i == "-.-.-.-")
j = '/';
if (i == "-....-")
j = '=';
if (i == ".-----.")
j = 39; //apostrophe
if (i == "-.-.-.-")
j = '(';
if (i == "-.-.-.-")
j = ')';
if (i == ".-.-.-.")
j = '&';
if (i == "...-.-.-")
j = '_';
if (i == ".-.-.-.-")
j = '"';
if (i == ".....-.-")
j = '$';
if (i == ".-.-.-.-")
j = '@';
if (j == ' ') //Si code non associé à un caractère
error = true;
tft.fillScreen(Display_Color_Black); //vidage de l'écran
tft.setCursor(0, 0); //Ecriture du "C:" suivi du code morse
tft.setTextColor(Display_Color_Green);
tft.print("C:");
tft.print(i);
tft.setCursor(0, 60); //En dessous, écriture du "P:" suivi de la phrase
tft.setTextColor(Display_Color_White);
tft.print("P:");
tft.print(phr);
if (error) //Si code inconnu
{
tft.setTextColor(Display_Color_Red); //Ecrire "?" rouge après la phrase
tft.print('?');
}
else //sinon
{
tft.setTextColor(Display_Color_Green); //Ecrire caractère reconnu dans "C:"
tft.print(j);
}
}

void setup()
{
Serial_JR6001.begin(9600); //Liaison série pour JR6001
pinMode(MatriceR, OUTPUT); //Gestion des LED
pinMode(MatriceV, OUTPUT);
pinMode(LedR, OUTPUT);
pinMode(LedV, OUTPUT);
digitalWrite(LedR, 1);
digitalWrite(LedV, 1);
pinMode(Busy, INPUT);
// Sets the MP3 player to loop tracks
Serial_JR6001.println("B4:02");
// Sets volume to 30
Serial_JR6001.println("AF:1");
// Utilisation de la mémoire FLAH du module
Serial_JR6001.println("A8:02");
delay(100);
// INIT écran TFT 2" (240x320) contrôleur ST7789

```

```

tft.init(240, 320); //Initialisaion de l'écran
tft.setRotation(3); // mise en mode paysage (0 à 3 pour chaque sens)
// initialise the display
tft.setFont(); //Premiers réglages
tft.fillScreen(Display_Color_Black);
tft.setTextColor(Display_Color_Red);
tft.setTextSize(3);
cap.begin(0x5B); //Initialisation MMP121
MPR(); //Premiere lecture MPR121
Morse(); //Premiere reconnaissance du morse pour afficher les caractères
}

void loop()
{
  while (touched == 0) //Tant que le MPR121 n'est pas touché
  {
    MPR();
    delay(50);
  }
  if (touched == 16) //Si contact effacer touché
  {
    if (i != "") //Si code morse non vide
    {
      i = ""; //Vider le code morse
      j = ' ';
    }
    else //Sinon
    {
      phr = phr.substring(0, phr.length() - 1); //Effacer la dernière lettre de la phrase
    }
  }
  else if (touched == 8) //Si contact trait touché
  {
    i = i + "-"; //Ajouter un trait au code morse
    digitalWrite(MatriceR, 1); //Allumer matrice rouge
    Serial_JR6001.print("A7:00003"); // Lire Klaxon long
  }
  else if (touched == 4) //Si contact point touché
  {
    i = i + "."; //Ajouter un point au code morse
    digitalWrite(MatriceV, 1); //Allumer matrice verte
    Serial_JR6001.print("A7:00001"); // Lire Klaxon court
  }
  else if (touched == 2) //Si contact espace touché
  {
    if (error) //Si erreur sur code morse
    {
      i = ""; //Effacer code morse pour le recommencer
    }
    else
    {
      if (j == '_') //Sinon Si code vide
        phr = phr + " "; //Insérer un espace à la place
      else //Sinon
      {
        phr = phr + j; //Ajouter caractère à la phrase
        i = "";
      }
    }
  }
  Morse(); //Actualiser code et phrase
  digitalWrite(MatriceR, 0); //Eteindre les matrices
  digitalWrite(MatriceV, 0);
  while (touched > 0) //Tant que contacts non relâchés
  {
    MPR(); //Lire le MPR121
    delay(50);
  }
}

```

Annexe 6

```
/* *****
 * CUBE_V3.c                mai-juin 2024
 * *****
 * MPR121 connecté sur 5 contacts de la sphère
 * Ecriture et décodage du nouveau moyen de communication
 * Affichage via communication Bluetooth
 * *****
#include <ArduinoBLE.h> //Ajout librairies
#include <Arduino_BMI270_BMM150.h>
#include "Adafruit_MPR121.h"
Adafruit_MPR121 cap = Adafruit_MPR121();
int touched;

#include <TensorFlowLite.h>
#include <tensorflow/lite/micro/all_ops_resolver.h>
#include <tensorflow/lite/micro/micro_error_reporter.h>
#include <tensorflow/lite/micro/micro_interpreter.h>
#include <tensorflow/lite/schema/schema_generated.h>
#include <tensorflow/lite/version.h>
#include "C:\Users\theod\OneDrive\cours\BUT GEII\Stage BUT2\Arduino ML\lecture_donnees\model.h"
//Ajout fichier d'en-tête pour reconnaissance de la reconnaissance de mouvements

BLEStringCharacteristic *Text = nullptr; //Utilisé pour initialiser le Bluetooth
BLEService *Service = nullptr;

const float accelerationThreshold = 1.5; // mesure si somme des accélérations dépasse cette valeur
const int numSamples = 119; //Nombre d'échantillons par mesure
int samplesRead = numSamples;

// global variables used for TensorFlow Lite (Micro)
tflite::MicroErrorReporter tflErrorReporter;

// pull in all the TFLM ops, you can remove this line and
// only pull in the TFLM ops you need, if would like to reduce
// the compiled size of the sketch.
tflite::AllOpsResolver tflOpsResolver;

const tflite::Model *tflModel = nullptr;
tflite::MicroInterpreter *tflInterpreter = nullptr;
TfLiteTensor *tflInputTensor = nullptr;
TfLiteTensor *tflOutputTensor = nullptr;

// Create a static memory buffer for TFLM, the size may need to
// be adjusted based on the model you are using
constexpr int tensorArenaSize = 8 * 2048;
byte tensorArena[tensorArenaSize] __attribute__((aligned(16)));

//Liste des gestes
const char *GESTURES[] = {
    "baisser",
    "droite",
    "lever",
    "tournis"};
#define NUM_GESTURES (sizeof(GESTURES) / sizeof(GESTURES[0]))

void MPR() //Lecture MPR avec sécurité identique à la V2
{
    touched = cap.touched();
    if (touched == 4095)
    {
        while (touched == 4095)
        {
            Text->writeValue("Erreur : Capteur de contact deconnecte !");
            touched = cap.touched();
            delay(1000);
        }
        cap.begin(0x5B);
        Text->writeValue("Erreur : Capteur de contact retrouve !");
    }
}

void setup()
{
    Service = new BLEService("180F"); //Lancement Bluetooth
    Text = new BLEStringCharacteristic("2A19", BLERead | BLENotify, 50);
    if (!IMU.begin() || !BLE.begin()) //Si l'IMU ou le Bluetooth ne démarre pas correctement
    {
```

```

    while (1); //Rester bloqué sur ne rien faire
}
// set advertised local name and service UUID:
BLE.setAdvertisedService(*Service); // add the service UUID
Service->addCharacteristic(*Text);
BLE.addService(*Service);
//Permet de nommer l'Arduino
BLE.setLocalName("LeCubeV3");
// set the initial value for the characteristic:
Text->writeValue("0");
// start advertising
BLE.advertise();

// get the TFL representation of the model byte array
tflModel = tfLite::GetModel(model);
if (tflModel->version() != TFLITE_SCHEMA_VERSION)
{
    Serial.println("Model schema mismatch!");
    while (1)
        ;
}
// Create an interpreter to run the model
tflInterpreter = new tfLite::MicroInterpreter(tflModel, tflOpsResolver, tensorArena, tensorArenaSize,
&tflErrorReporter);

// Allocate memory for the model's input and output tensors
tflInterpreter->AllocateTensors();

// Get pointers for the model's input and output tensors
tflInputTensor = tflInterpreter->input(0);
tflOutputTensor = tflInterpreter->output(0);

cap.begin(0x5B); //Initialiser le MPR121
MPR(); //Première mesure du MPR121
}

// String Level_String;
float aX, aY, aZ, gX, gY, gZ; //Mesures échantillons

String Texte = "", mot = ""; //Texte utilisé pour former des phrases à afficher au terminal
char lettre = ' '; //mot et lettre utilisés pour l'écriture
char lettres1[9] = {'a', 'e', 'i', 'o', 'u', 'y', 'w', 'x', 'z'}; //Tableaux des caractères
char lettres2[8] = {'b', 'c', 'd', 'f', 'g', 'h', 'j', 'k'};
char lettres3[9] = {'l', 'm', 'n', 'p', 'q', 'r', 's', 't', 'v'};
int nombre = 0, dernier = 0; //Compteur d'appuis successifs et retenue du dernier contact touché
boolean active = false; //Etat vrai quand mesure en cours

void loop()
{
    // listen for BLE peripherals to connect:
    BLEDevice central = BLE.central();
    if (central)
    {
        while (central.connected()) //Tant que connecté au Bluetooth
        {
            //Si pas de mesure en cours
            if (active == false && samplesRead == numSamples)
            {
                MPR(); //Lire le MPR121
                if (touched != 0) //Si contact touché
                {
                    switch (touched) //Réaction le capteur touché
                    {
                        case 16: // Pouce
                            dernier = 0;
                            break;
                        case 8: //Index
                            if (dernier == 1) //Si index déjà touché juste avant
                            {
                                nombre++; //Incrémenter le compteur
                                if (nombre == 10) //Réinitialiser le compteur si besoin
                                {
                                    nombre = 1;
                                }
                            }
                            else //Si index non touché juste avant
                            {
                                dernier = 1; //Mettre à jour le dernier contact touché
                                nombre = 1; //Recommencer le compteur
                            }
                        }
                    }
                }
            }
        }
    }
}

```



```

        lettre = lettres1[nombre - 1]; //Modifier la lettre selon le compteur
        break;
    case 4: //Majeur, même fonctionnement que l'index
        if (dernier == 2)
        {
            nombre++;
            if (nombre == 9)
            {
                nombre = 1;
            }
        }
        else
        {
            dernier = 2;
            nombre = 1;
        }
        lettre = lettres2[nombre - 1];
        break;
    case 2: //Annulaire, même fonctionnement que l'index
        if (dernier == 3)
        {
            nombre++;
            if (nombre == 10)
            {
                nombre = 1;
            }
        }
        else
        {
            dernier = 3;
            nombre = 1;
        }
        lettre = lettres3[nombre - 1];
        break;
    case 1: // Auriculaire
        dernier = 4;
        break;
    default: //Informer d'un contact multiple
        Text->writeValue("Plusieurs");
    }
    if (dernier == 0) //Si pouce touché, passer à la lettre suivante
    {
        mot = mot + lettre;
        lettre = ' '; //Permettra d'insérer un espace si contact espace touché à nouveau
        Texte = "Phrase : " + mot + "."; //Construction de la phrase
        Text->writeValue(Texte); //Affichage de la phrase
    }
    else if (dernier == 4) //Sinon si auriculairz touché, supprimer le derier caractère
    {
        mot = mot.substring(0, mot.length() - 1);
        lettre = ' ';
        Texte = "Phrase : " + mot + ".";
        Text->writeValue(Texte);
    }
    else //Sinon, afficher le nouveau caractère
    {
        Texte = "lettre : " + (String)lettre;
        Text->writeValue(Texte);
    }
    while (touched != 0) //Tant que contacts touchés
    {
        MPR(); //Mesurer le MPR121 à nouveau
        delay(50);
    }
}
if (IMU.accelerationAvailable()) //Si IMU disponible
{
    IMU.readAcceleration(aX, aY, aZ); //Lire les accélérations
    float aSum = fabs(aX) + fabs(aY) + fabs(aZ); //Calculer la somme des accélérations
    if (aSum >= accelerationThreshold) //Si cette somme est supérieure à la limite
    { //Préparer et indiquer le lancement de la mesure
        samplesRead = 0;
        Text->writeValue("Mesure");
        active = true;
    }
}
}
// check if the all the required samples have been read since
// the last time the significant motion was detected
if (active == true && samplesRead < numSamples)

```

