El Kazdadi Moad Pitot-Belin Hugo Groupe 2
Jacquart Maxime Chevrier Lauryn

Apprentissage de la programmation
Rapport technique

La programmation est un domaine de plus en plus exploité de nos jours et fait appel à une logique et un état d'esprit qui lui sont propres. Ainsi, le fait de s'y initier dès le plus jeune âge fournit un atout considérable à tous les futurs programmeurs. Dans cette optique, notre équipe a décidé de développer un module d'apprentissage de la programmation afin de permettre aux plus jeunes de manipuler une interface numérique et de voir de manière directe les conséquences de leurs consignes sur un robot réel pouvant se déplacer sur un damier dans quatre directions.

Nos objectifs sont donc de proposer une interface ludique et accessible à des enfants de 10 ans, de proposer plusieurs modes de jeu et de disposer d'un robot ayant des mouvements rapides et fiables (moins de 20 secondes par mouvement et une erreur de moins de 2 mm en x et en y sur la position).

Sommaire

➢ Interface Matlab

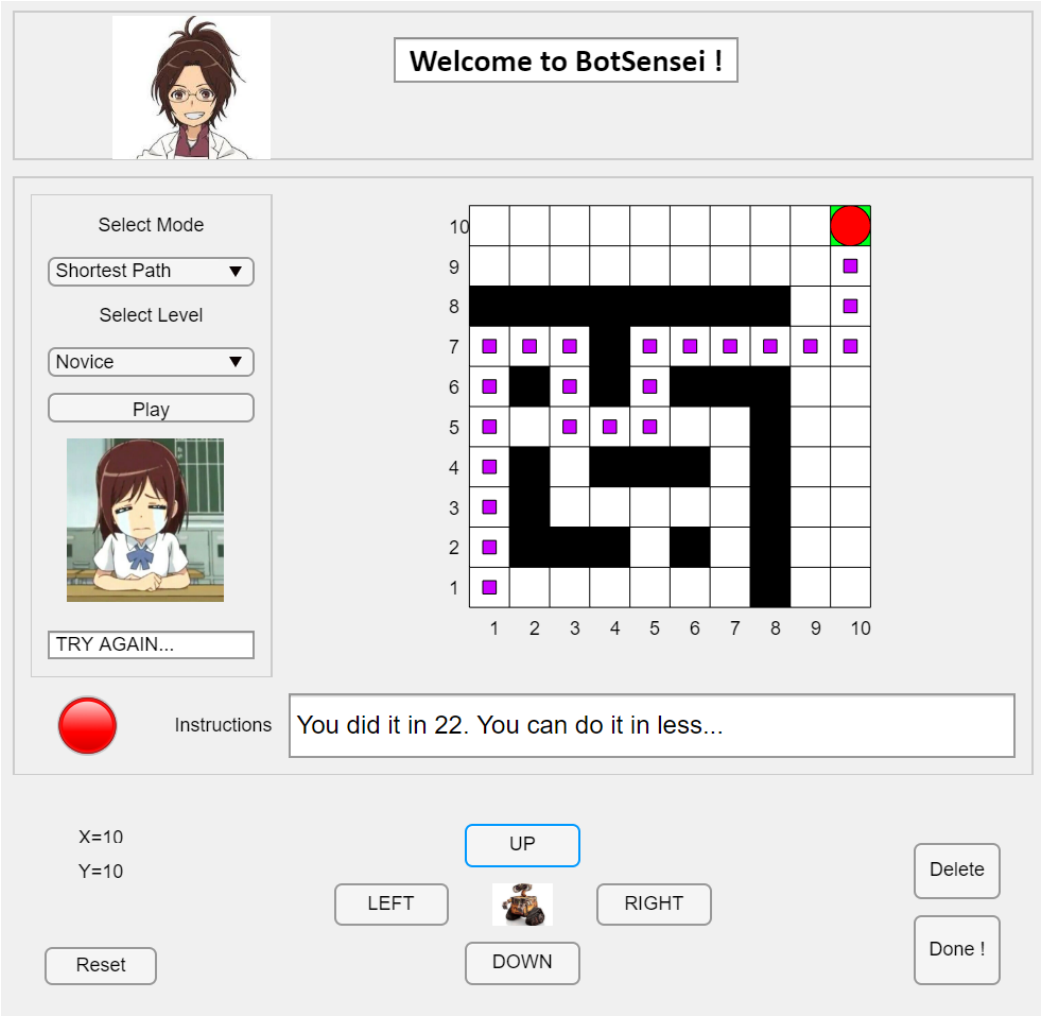
Il est à noter que les explications des fonctions utilisées sont indiquées en commentaire dans le programme lui-même fourni en annexe. Les explications qui suivent portent donc principalement sur la structure de l'interface et la façon dont il faut interagir avec l'interface, bien que cette dernière reste très instinctive. Pour bien comprendre le fonctionnement du programme, il est donc indispensable de compléter la lecture de cette partie par la lecture du code Matlab commenté.

De plus, l'application que nous avons utilisée pour ce travail est l'App Designer de Matlab, qui nous permet d'avoir accès à la fois à un éditeur de code et à un éditeur visuel. Certains codes de déclaration d'objets (comme un bouton, une lampe, ou un menu déroulant) ont donc été générés automatiquement par l'App Designer à partir de l'éditeur visuel. Cette partie là du programme n'est donc pas commentée, car il nous est impossible d'accéder à cette partie, qui n'est que visible et non modifiable.

El Kazdadi Moad Pitot-Belin Hugo Groupe 2
Jacquart Maxime Chevrier Lauryn

L'interface Matlab est le centre du système interactif. Il permet de relayer les commandes de l'utilisateur au microcontrôleur, qui lui-même envoie ensuite les informations aux actionneurs. Il sert à choisir le mode de jeu et le niveau, ainsi qu'à actionner les diverses commandes permettant de commander le robot.

Voici l'allure finale de notre interface :



L'interface présente les fonctionnalités suivantes :

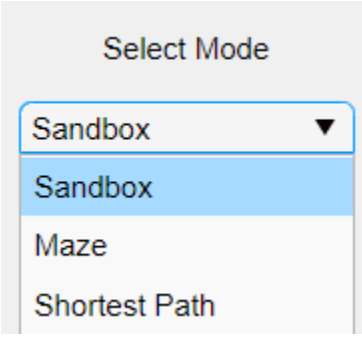
- Choix du mode de jeu
- Choix du niveau de jeu
- Le bouton "Play" : lancer une partie
- Les touches directionnelles
- Le bouton "Delete", un ctrl+Z intégré
- Le bouton "Reset", pour relancer une partie
- Le bouton "Done", pour communiquer avec le micro-contrôleur
- L'affichage en cas de victoire ou défaite

El Kazdadi Moad Pitot-Belin Hugo Groupe 2
Jacquart Maxime Chevrier Lauryn

★ Le choix du mode de jeu

Notre système prévoit trois modes de jeu, à choisir dans un menu déroulant : Sandbox, Maze et Shortest Path.

- **Sandbox :** Ce mode permet de se familiariser avec les touches directionnelles et les autres commandes accessibles à l'utilisateur. La grille dans laquelle se déplace le robot est donc vide et ne contient pas d'obstacles.
- **Maze** : Dans ce mode, la grille générée contient des obstacles. Le robot est alors placé dans une position initiale et doit se déplacer jusqu'à la case d'arrivée, de couleur verte.
- **Shortest Path :** Ce mode ressemble au précédent, à ceci près qu'il faut trouver le chemin le plus court permettant d'arriver à la position finale. Si l'utilisateur réussit, un message de succès apparaît ; sinon, l'utilisateur est invité à réessayer.

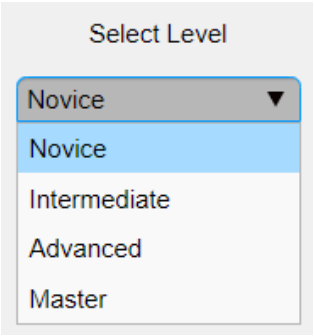


Le choix de l'utilisateur est retenu à l'aide des booléens *app.checkSandbox, app.checkMaze* et *app.checkPath.* Par exemple, l'utilisateur choisit le mode "Maze" dans le menu déroulant. Lorsque l'utilisateur appuie sur le bouton "Play", la valeur 1 est affectée au booléen *app.checkMaze* tandis que la valeur 0 est affectée aux deux autres. Ces booléens sont ensuite réutilisés dans d'autres fonctions du programme en tant que conditions de test, pour n'afficher que les fonctionnalités correspondantes au mode de jeu "Maze".

★ Le choix du niveau de jeu

Une fois le mode de jeu choisi, différents niveaux sont disponibles : Novice, Intermediate, Advanced, Master.

Il est à noter qu'en raison d'un manque d'expertise en level design, les niveaux actuellement contenus dans le code ne correspondent pas aux niveaux de difficulté annoncés. De plus, nous avons pour commencer travaillé avec des matrices de taille 10x10. Il est évident que des matrices 100x100 seraient plus adaptées pour un tel jeu. Cependant, à condition d'avoir accès à des matrices de taille 100x100 déjà réalisées, ce n'est pas difficile à implémenter dans notre code actuel. Il suffit de modifier la valeur de CaseNombre, ligne 10, et de modifier en fonction la matrice app.obstacles*.*



Le choix du niveau combiné au choix du mode de jeu modifie la constitution de la matrice *app.obstacles*, matrice constituée de 1

(case avec obstacle) et de 0 (case vide), et qui représente la grille (ou damier) affichée sur l'interface. Par exemple, dans le mode "Sandbox", *app.obstacles = zeros(CaseNombre)*, étant donné qu'il s'agit d'une grille intégralement vide.

★ Le bouton Play

Une fois le mode de jeu et le niveau choisis, le bouton "Play" permet la mise en place du niveau : la fonction *PlayButtonPushed()* affiche la matrice d'obstacles, le robot (un rond rouge) dans sa position initiale et la position d'arrivée (une case verte) qui ont été déterminées au moment de la sélection du niveau.
Lorsque le bouton "Play" est actionné, tous les booléens sont remis à zéro pour prendre en compte le démarrage d'un nouveau niveau.
La fonction *afficherGrille()* est sollicitée afin d'afficher la grille correspondant à *app.obstacles*.
Enfin, la fonction *PlayButtonPushed()* s'assure également que tous les éditeurs de texte soient vides et que les images temporaires ne soient plus affichées.

Le bouton Play, après affichage du niveau, envoie également les commandes de position initiale selon X et selon Y au micro-contrôleur. Il suffit de transformer les deux entiers entre 1 et 10 en une chaîne d'un unique caractère (c'est pour cette raison que l'entier 10 est représenté par la chaîne '0'), puis de les envoyer à la Nucléo. L'envoi est effectué de la même manière que pour le bouton "Done", donc expliqué ci-après.

★ Les touches directionnelles

Ce sont elles qui permettent de déplacer le robot sur la grille. La fonction vérifie que le robot ne va ni sur un obstacle ni sur un bord de la grille avant de le déplacer. Après chaque déplacement, on vérifie si le robot est sur la case d'arrivée. Si c'est le cas, on modifie le booléen *app.checkMazeDone* ou *app.chekPathDone* en fonction du mode choisi. Celui-ci permettra d'activer les messages de réussite ou de défaite (si on n'a pas utilisé le chemin le plus court).

Si la case Parcours est cochée, des carrés violets seront affichés sur les cases où est passé le robot.

En plus du déplacement du robot, on rentre la commande de l'utilisateur dans la chaîne d'instructions *app.instructions* qui servira par la suite à envoyer les commandes à la carte Nucléo. On affiche également cette instruction dans l'éditeur de texte prévu à cet effet.

★ Le bouton Delete

Ce bouton est conçu pour faire office de ctrl+Z. Son principe de fonctionnement est simple : il regarde la dernière instruction donnée par l'utilisateur et effectue la commande 'inverse'. Il s'assure également de retirer la dernière instruction de l'éditeur de texte correspondant et de la chaîne d'instructions *app.instructions.*

Il est à noter que ce bouton ne peut être activé que si l'utilisateur a déjà effectué au moins une instruction, et si le niveau n'est pas déjà terminé, soit si le robot n'est pas arrivé sur la case verte. D'où le test en début de la fonction DeleteButtonPushed() *:* if not(isempty(app.instructions)) && not(app.checkMazeDone) && not(app.checkPathDone) && not(app.checkPathTooSoon).

★ Le bouton Reset

Ce bouton sert à réinitialiser le niveau en cours. Pour ce faire, il replace le robot en position *app.RobotXIni* et *app.RobotYIni*. Puis il ré-affiche la grille en partant de cette nouvelle position afin de faire disparaître les carrés bleus du parcours et réafficher le robot sur la position initiale.

Tous les éditeurs de texte sont également vidés, la lampe reprend une couleur blanche, l'image de victoire ou défaite disparaît, et tous les booléens de fin de niveau ou de niveau en cours sont remis à 0.

★ Le bouton Done

Ce bouton est l'un des deux seuls boutons avec le bouton "Play" qui communiquent directement avec le micro-contrôleur. Après activation, la chaîne d'instructions *app.instructions* est envoyée à la carte Nucléo. Cet envoi se fait caractère par caractère, en attendant entre chaque caractère un signal de la Nucléo indiquant qu'elle est prête à recevoir l'instruction suivante.

En mode de jeu "Shortest Path", ce bouton est prévu pour n'être activé qu'après avoir réalisé le niveau. En effet, c'est l'intégralité de la chaîne *app.instructions* qui est envoyée au micro-contrôleur, chaîne qui doit rester complète pendant tout le niveau pour pouvoir activer la fonction *PathFinding()*. Ainsi, il n'est pas encore à ce stade possible d'envoyer la première partie des instructions, puis d'envoyer la deuxième partie. Appuyer sur le bouton "Done" une deuxième fois à mi-parcours enverrait la chaîne complète à la Nucléo, et le robot risque donc de rentrer dans un obstacle.

C'est pourquoi, après activation de ce bouton dans ce mode de jeu, on empêche le robot de bouger à l'aide du booléen *app.checkPathTooSoon.*

En revanche, en modes de jeu "Sandbox" et "Maze", ce problème ne se pose pas. Il faut cependant veiller à vider l'éditeur de texte d'instructions et la chaîne d'instructions, afin de ne pas renvoyer à nouveau les anciennes instructions en plus des nouvelles.

★ L'affichage en cas de victoire ou défaite

- Maze : dans la fonction *deplacerRobot* (appelée à chaque déplacement) , on vérifie à la fin si le robot est sur la case d'arrivée. Si c'est le cas, on affiche une image, un message de victoire et on allume la lampe en vert.

- ShortestPath : au début de chaque niveau, on calcule dans la fonction *PathFinding()* la longueur du chemin le plus court possible à l'aide de la méthode de Dijkstra.
  De même que précédemment, on vérifie à chaque déplacement si le robot est sur la case d'arrivée. Si c'est le cas, on compare la longueur du parcours effectué avec celle du chemin le plus court estimée plus tôt. Si elles sont égales, on affiche les messages et images de succès dans la fonction *DisplayPathDone()*. Sinon, on affiche dans cette même fonction une image de défaite et un message informant qu'il est possible de trouver plus court. Dans les deux cas, l'utilisateur est informé dans le message du nombre de déplacements effectués.

➢ Microcontrôleur

La carte Nucleo joue le rôle de relais entre l'interface MATLAB sur la machine et les composantes électroniques des actionneurs. Le code contient 3 parties importantes :

- Le contrôle précis du courant pilotant les 8 bobines dans les 2 moteurs
- Le mouvement sur la grille
- La communication avec l'interface

★ Contrôle des moteurs

Les moteurs utilisés pour se déplacer sont des moteurs pas à pas. Pour les piloter il faut une séquence de courants sur les 4 bobines afin d'appliquer un champ magnétique tournant sur le rotor. La fonction *reglage_sigs* prend en variable 4 signaux et une dimension (0 pour horizontal et 1 pour vertical) et permet d'imposer une tension sur les 4 bobines du moteur visé (0 pour 0 V et 1 pour 3.3 V).

On peut donc appliquer cette fonction 4 fois avec des signaux différents sur les bobines afin d'effectuer un tour entier du moteur. La fonction *prochainStep()* prend en variable une dimension et un nombre entier entre 0 et 3, et permet d'appliquer 4 tensions de commande pour mettre le moteur dans l'une des 4 position cardinales :

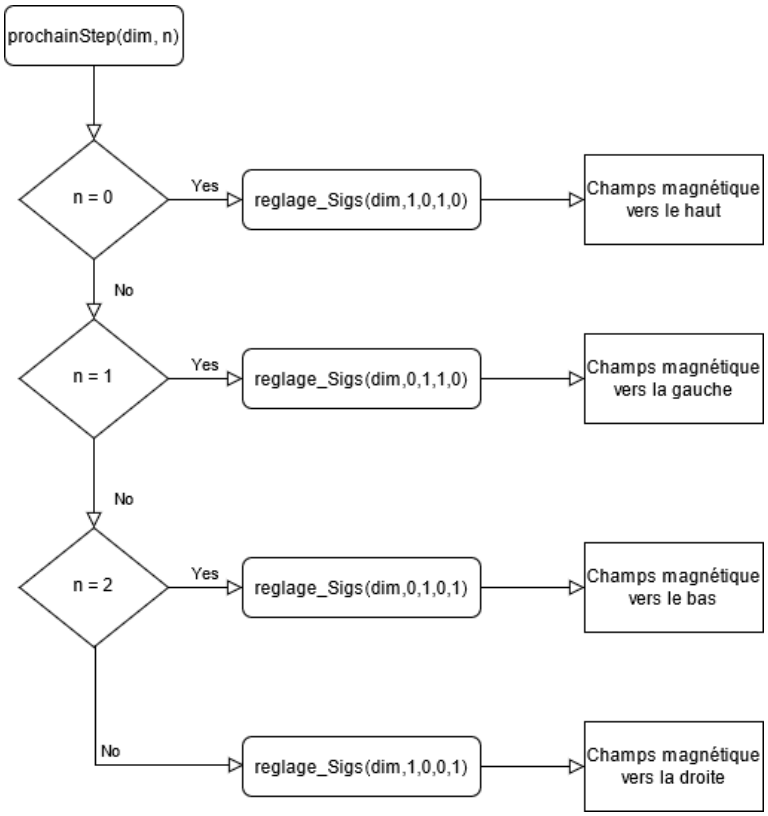


Figure : Schéma bloc du contrôle des moteurs

★ Déplacement sur la Grille

Pour faire tourner le moteur d'un cycle entier, on applique la fonction *prochainStep()*, dans une boucle de façon à ce que le routeur se mette en position haut, puis gauche, puis bas, puis droite et puis haut encore pour revenir à la position initiale. Et on répète cette opération autant de fois qu'on souhaite faire de pas. Après chaque application de *prochainStep()*, on vérifie que les capteurs de la direction de déplacement ne sont pas actionnés, au cas où la fonction est terminée.

Les fonctions *marche_up(), marche_down(), marche_right() et marche_left()* prennent en variable un nombre entier (n) et un réel (t). Elle effectue n pas du moteur en attendant t secondes entre deux positions consécutives. Elle permet donc de faire se déplacer dans le cadre et de régler la vitesse en diminuant le temps de pause.

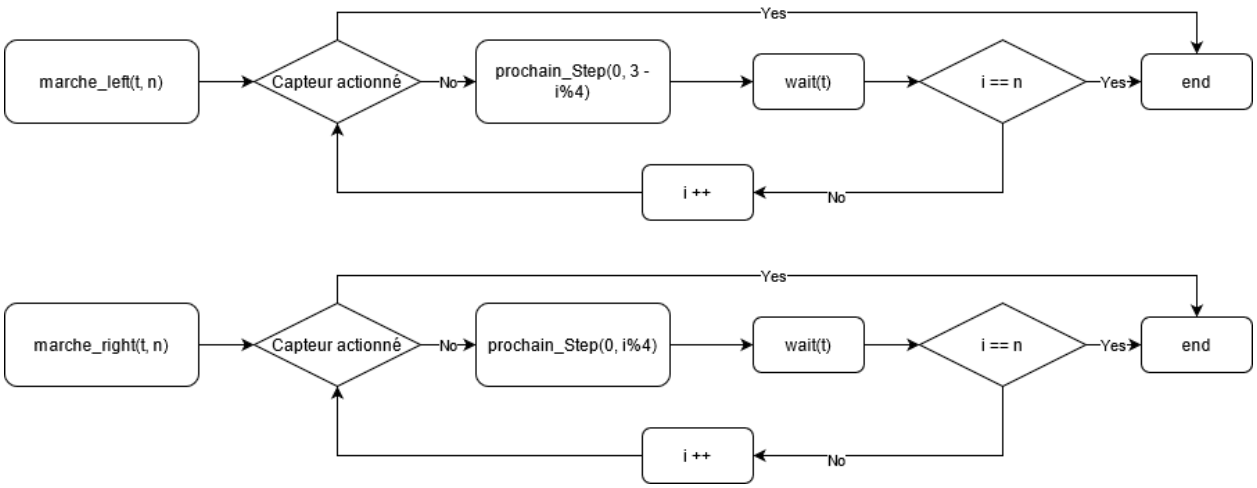


Figure : Schéma bloc pour la marche à gauche et à droite

Remarque : Pour la marche dans le sens positif (droite), la variable de position dans *prochainStep()* est i%4 pour appliquer la séquence de 4 commandes périodiquement. Mais pour la marche dans l'autre sens, la variable est à 3 - i%4 car on souhaite avoir la même période mais dans le sens inverse.

Pour se déplacer exactement d'une case sur la grille du jeu, on mesure les grandeurs *pas_hor* et *pas_ver* correspondant au nombre de pas nécessaire pour parcourir l'entièreté de la grille (10 cases) verticalement et horizontalement (les deux moteurs ne sont pas calibrés de la même façon). Et donc pour avancer d'une seule case on applique la commande de marche avec le nombre de pas égale à *pas_hor*/9 (ou *pas_ver*/9).

La période d'attente entre deux pas est mise à 5 ms pour un temps de parcours de la grille (diagonalement) de 15 secondes.

Pour initialiser le système, il faut le mettre dans une position connue. Pour ceci, on utilise les capteurs placés sur les bords en bas et à droite. Le système effectue des pas en boucle vers la droite jusqu'à ce que le capteur s'active, et ensuite la même opération est effectuée vers le bas. A la fin de l'initialisation, on est sûr d'être en position en bas à droite.

★ Communication avec l'Interface

L'échange d'information entre la carte Nucleo et l'interface MATLAB se fait à travers une liaison série RS232 à 115200 bauds.
La carte Nucleo reste en attente d'une commande sous la forme d'un caractère envoyé à travers la liaison. Cette commande est ensuite effectuée par le système, et une lettre 'n' est envoyée pour informer l'interface que l'opération est terminée. Les commandes possibles sont les suivantes:

- 'u' : Le système traverse une case vers le haut. La commande est stockée dans la liste d'instructions effectuée

- ‘d’ : Le système traverse une case vers le bas. La commande est stockée dans la liste d’instructions effectuée
- ‘r’ : Le système traverse une case vers la droite. La commande est stockée dans la liste d’instructions effectuée
- ‘l’ : Le système traverse une case vers la gauche. La commande est stockée dans la liste d’instructions effectuée
- ‘z’ : Le système lis la liste d’instructions précédentes et effectue les pas inverse afin de revenir à sa position initiale
- ‘i’ : Le système réinitialise.

★ Défis rencontrés

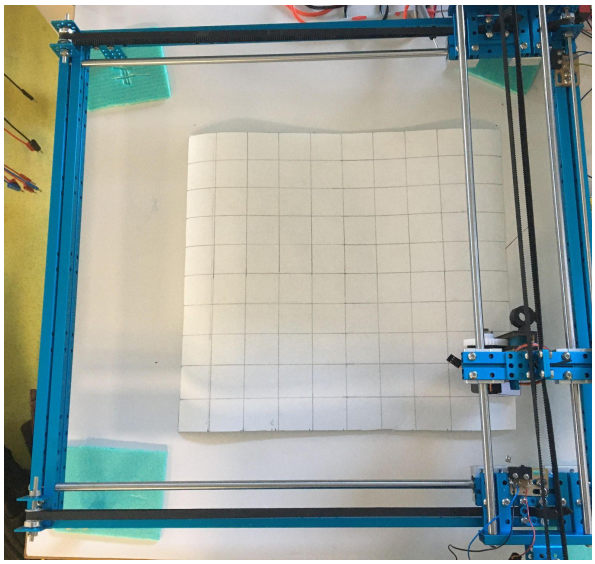
On a eu beaucoup de difficultés pendant la phase de débogage. On s’est rendu compte après 10h de travail que certaines entrées sur la carte Nucléo étaient défectives et qu’il fallait changer de carte, ce qui a résolu la majorité des soucis.

Une fonctionnalité que nous n’avons pas eu le temps d’implémenter est l’initialisation avancée sur la grille. Cette fonction est censée recevoir de l’interface Matlab une commande d’initialisation, suivie d’un couple de coordonnées indiquant la position voulue. Mais pour des raisons que nous n’avons pas pu identifier, le code C++ affiche une erreur après la réception de la commande d'initialisation. On a fini par abandonner cette fonction mais il est possible avec un peu plus de temps d’identifier le problème et éventuellement le résoudre.

➢ Système électrique

★ Globalité mécanique

Le système physique est un cadre métallique assemblée, un robot conçu pour se déplacer sur un cadre de 20*20 cm.



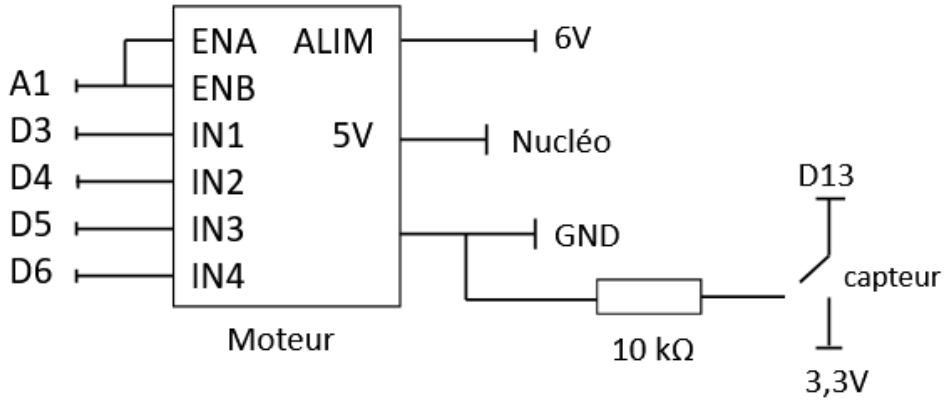
El Kazdadi Moad Pitot-Belin Hugo Groupe 2
Jacquart Maxime Chevrier Lauryn

★ Composition

- 4 barres en acier bleu créant le cadre de la machine
- Une tige en acier soutenant le moteur horizontal
- 2 moteurs à rotation avec chacun une courroie de transmission en caoutchouc
- 2 capteurs d'impact sur les extrémités horizontales et verticales.
- Alimentation électrique 3,3V et 6V
- Carte nucléo

Un Moteur fonctionne avec 4 bobines autour d'un aimant relié au cylindre en contact avec la courroie. L'aimant tourne pour s'aligner avec les champs magnétiques des bobines. Les entrées IN1, IN2, IN3 et IN4 correspondent respectivement aux bobines du dessus, de la droite, du bas et de la gauche.

★ Circuit Électrique



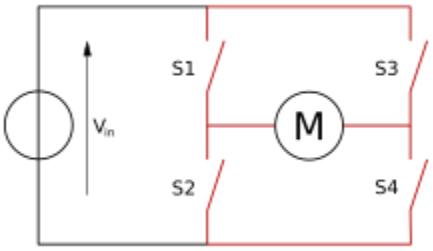
Chaque moteur comporte :

- Une entrée GND(Ground) connecté à la nucléo
- Une entrée 5V connecté à la nucléo
- Une entrée ALIM connecté à l'alim 6V
- Deux entrée Enabler connecté à l'A1 de la nucléo lorsque les moteurs doivent s'actionner
- 4 entrées IN pour recevoir les instructions de rotations

Hors moteur, le capteur est relié au Ground si il est inactif et à 3,3 V si il est actif et rejoint l'entrée D13.
Les moteurs et les capteurs fonctionnent indépendamment et le tout est basé sur un cadrillage de cases blanche avec des jetons noir pous simuler les obstacles. Un simple scotch peut ancrer le quadrillage.

El Kazdadi Moad Pitot-Belin Hugo Groupe 2
Jacquart Maxime Chevrier Lauryn

Le convertisseur qui alimente le moteur contient un pont H pour inverser le sens de déplacement. Le pont fonctionne de la manière suivante:



Combinaisons d'états des commutateurs

État des commutateurs				Résultat à la charge
S1	S2	S3	S4	
✗				Aucune tension aux bornes de la charge.
✓	✗		✓	Courant positif à travers la charge.
✗	✓		✗	Courant négatif à travers la charge.
✓		✗		Charge court-circuitée.
✗		✓		

★ Problèmes rencontrés

- Réaliser un tel montage requiert de nombreux câbles ce qui cause certaines perturbations parfois des court-circuits. Les récepteurs des Nucléo sont très fragiles et ont tendance à s'user après l'utilisation.
- Le moteur vertical subissait un problème d'enraillement, oscillant sur sa position d'équilibre et vibrant bruyamment.
- Les obstacles de grandes tailles empêchent une bonne réinitialisation. On utilise donc des pièces d'othello pour simuler des zones inaccessibles.

➢ Analyse des résultats

Le prototype construit à la fin du projet laisse place à un nombre d'améliorations, mais valide les demandes spécifiées dans le cahier de charge :

- Le système est suffisamment rapide pour traverser l'entièreté de la grille en diagonale en 15 secondes. Les moteurs utilisés sont capables de tourner encore plus vite, jusqu'à une vitesse permettant de parcourir la grille en moins de 10 secondes, mais on a choisi de les garder suffisamment lents pour que l'enfant en apprentissage puisse éventuellement suivre les mouvements et les comparer à ses commandes.

- Quand le système effectue un déplacement partant du centre d'une case, il retombe dans le centre de sa case cible avec une erreur de l'ordre de 1 mm. Cette erreur est principalement due à une incertitude sur la mesure du facteur de conversion entre les pas effectués par un moteur, et la distance qu'il parcourt. On atteint donc pas la limite de précision ultime des moteurs (fixées par le pas), mais on est suffisamment précis pour satisfaire le cahier de charges.

- Le design de l'interface a comme priorité d'être clair et intuitif. Afin de la tester, nous avons demandé à quelques personnes de tester l'application avec un minimum d'instructions. Leurs retours ont été utilisés pour améliorer l'interface pour la rendre utilisable sans besoin d'explication.

Pour illustrer le fonctionnement du prototype, une vidéo à été prise du mode "Shortest Path"

➢ Retour d'expérience

Lors de la première séance, nous avons décidé de séparer le travail comme suit : interface à coder en Matlab ou en Python, codage du programme pour le micro-contrôleur Nucléo, et câblage du montage électronique.

Maxime et Moad ont respectivement travaillé sur les parties codage micro-contrôleur et câblage. Lauryn a travaillé uniquement sur l'interface Matlab, et notamment les détails finaux tels que l'affichage des images ou textes, la gestion des booléens et l'algorithme de pathfinding. Hugo a travaillé en majorité sur l'interface en binôme avec Lauryn, puis s'est concentré durant les deux dernières séances sur les tests physiques dans le labyrinthe "grandeur nature" et a assuré la liaison entre les groupes.

Pour ce qui est de l'interface, nous avons rapidement choisi de travailler avec l'App Designer de Matlab. M. Villemejane nous a aidé à démarrer avec un programme initial affichant un damier, le robot et le parcours suivi par ce dernier. Un lien vers ce programme est fourni dans les annexes.

Le développement d'une telle interface nous a principalement permis de nous familiariser avec la programmation objet en langage Matlab. Il s'agit de plus d'une interface Homme-Machine, nous avons donc dû apprendre à gérer les actions déclenchées à la suite d'une commande donnée par l'utilisateur.

Les pôles Interface et Micro-contrôleur ont pu également se re-familiariser avec l'échange de données entre un micro-contrôleur embarqué et une interface PC au travers du protocole RS232.

Pour le câblage, cette expérience a permis de comprendre le fonctionnement d'un moteur pas-à-pas, ainsi que d'un pont en H. Nous avons également appris à réaliser des connecteurs avec un fer à souder.

El Kazdadi Moad Pitot-Belin Hugo Groupe 2
Jacquart Maxime Chevrier Lauryn

Chacun s'occupant d'une partie entièrement différente, nous avons dû réaliser un travail en équipe efficace afin de coordonner les avancées de chacun et d'assurer une bonne communication entre les différents pôles. Nous n'avons éprouvé aucune difficulté à ce niveau et avons su relayer les informations suffisantes à chacun pour obtenir un résultat final satisfaisant.

➢ Annexes

- **code Matlab et images, code C++**

BotSensei.mlapp : Programme final Matlab
Rectangle_sur_figure.mlapp : Programme initial fourni par M. Villemejane
Main.ccp : Programme embarqué