Livrable

Projet IETI : Table Traçante

Printemps 2021

Baptiste Bouhet

Thadek Ferrand

Étienne Minnaert

Xinxin XU

Table des matières

Références

Table traçante : XY-Plotter Robot Kit v2.0 (With electronic) – ORBIT ELECTRONIC (orbit-dz.com)

Site du Lense : http://lense.institutoptique.fr/

Épicycloïde — Wikipédia (wikipedia.org)

mickey curve - Wolfram|Alpha (wolframalpha.com)

http://nounoudunord.centerblog.net/2007-coloriage-canard-dessine-par-nounoudunord

Introduction

L'objectif de ce projet électronique 1A a été de programmer la table traçante en deux dimensions XY-Plotter. Cette table traçante fonctionne à l'aide de deux moteurs pas à pas indépendants selon les directions X et Y. Un programme est compilé sur le logiciel Mbed puis est envoyé sur une carte Nucléo.

La première étape a été de faire fonctionner ces deux moteurs avec des boutons poussoirs, puis d'automatiser les déplacements d'inclinaisons quelconques selon un schéma préétabli avant la compilation. Une fois ces étapes réalisées, il a été possible de tracer des figures géométriques simples (carrés, triangle), puis toute sorte de courbes paramétriques (cercles, épicycloïdes, Mickey Mouse) et de tracer un dessin à la main sur l'ordinateur qui sera ensuite reproduit par la table traçante.

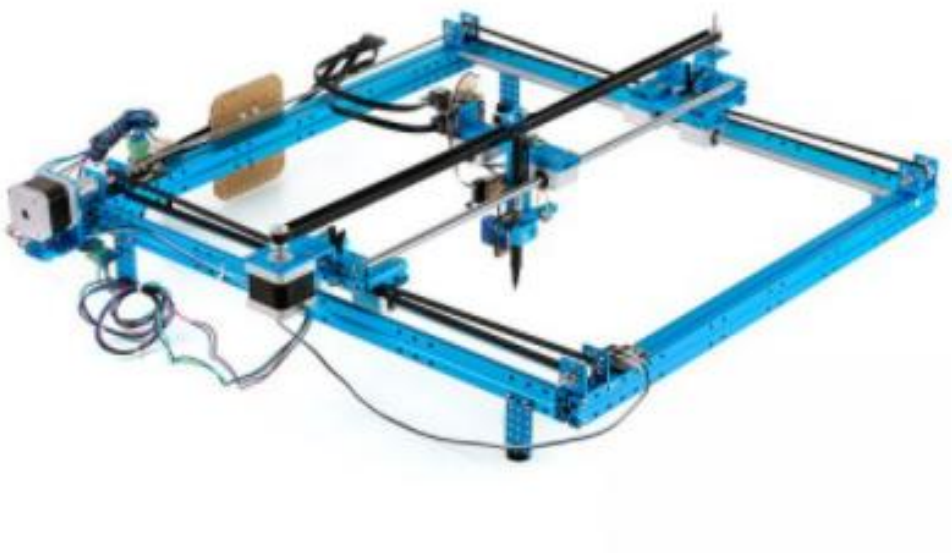


Figure 1 : Table traçante

Découpage fonctionnel

Liste des fonctionnalités réalisées :

➢ Moteurs pas à pas fonctionnants :
- En continu : pilotage automatisé des moteurs
- Avec des boutons marche-arrêt

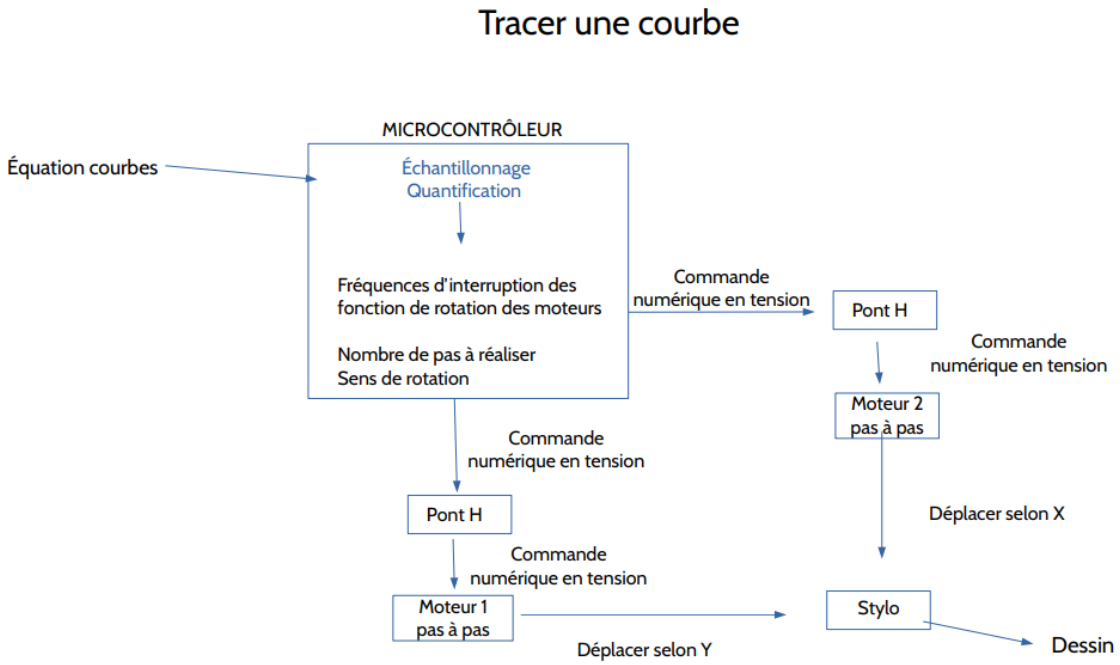
➢ Table traçante permettant de tracer :
- Dans les huit directions cardinales à vitesse constante avec un joystick (représenté par quatre boutons)
- Dessins composés de multiples segments de droites
- Figures géométriques simples : rectangles, triangles particuliers
- Des cercles (pas très réguliers)
- Toutes les courbes sans levé du stylo notamment des courbes cycloïdales et Mickey

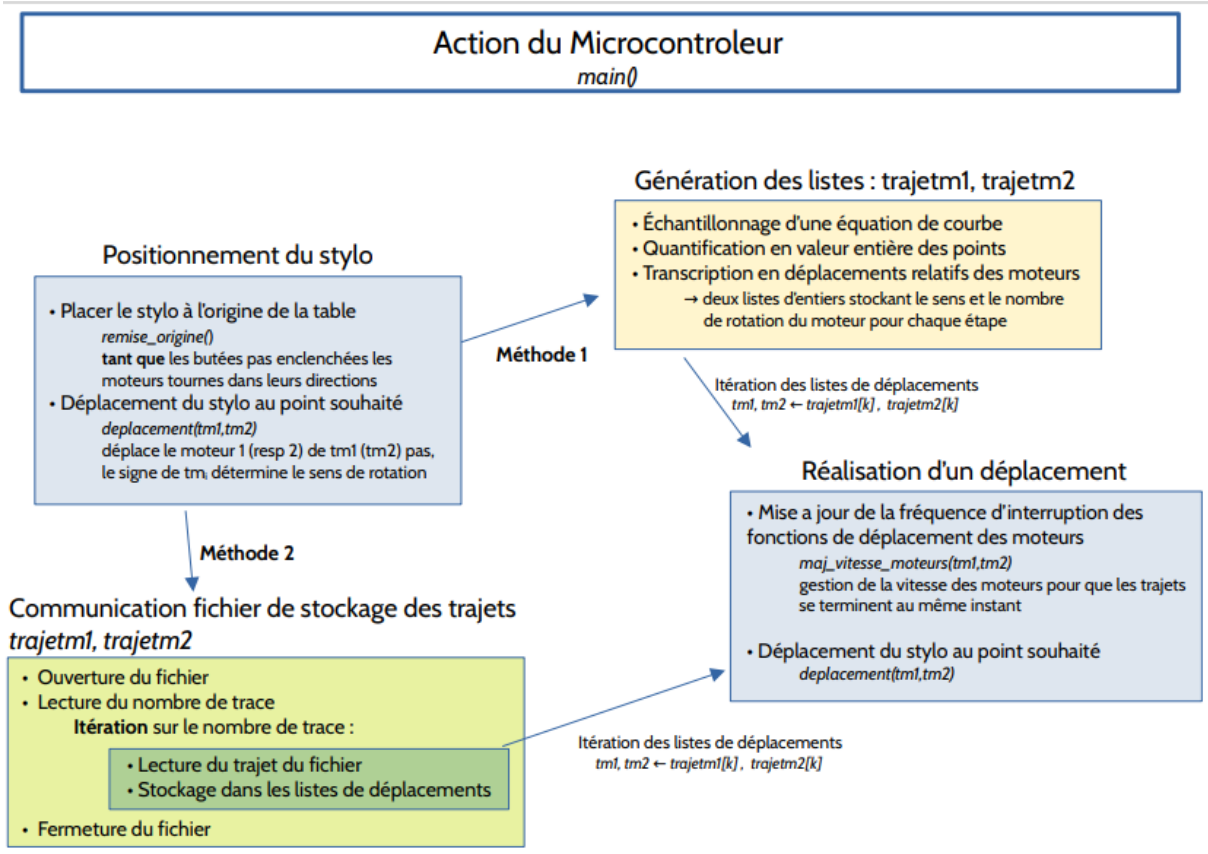
➢ Fonctions de déplacements du stylo sur la table :
- Remise à l'origine de la table
- Déplacement en ligne droite à un point précis par rapport à une référence (origine ou point précédent)

➢ Interfaces graphiques pour la réalisation d'une figure (à titre indicatif) :
- Algorithmes de traitement d'image (redimensionnement, seuillages pour le détourage)
- Réalisation d'une interface pour dessiner librement ou détourer des images
- Stockage de l'information sur un fichier texte

- Algorithme de décryptage d'image SVG simple
- Conversion d'information d'image SVG en déplacement des moteurs

Schémas blocs





Schémas électriques

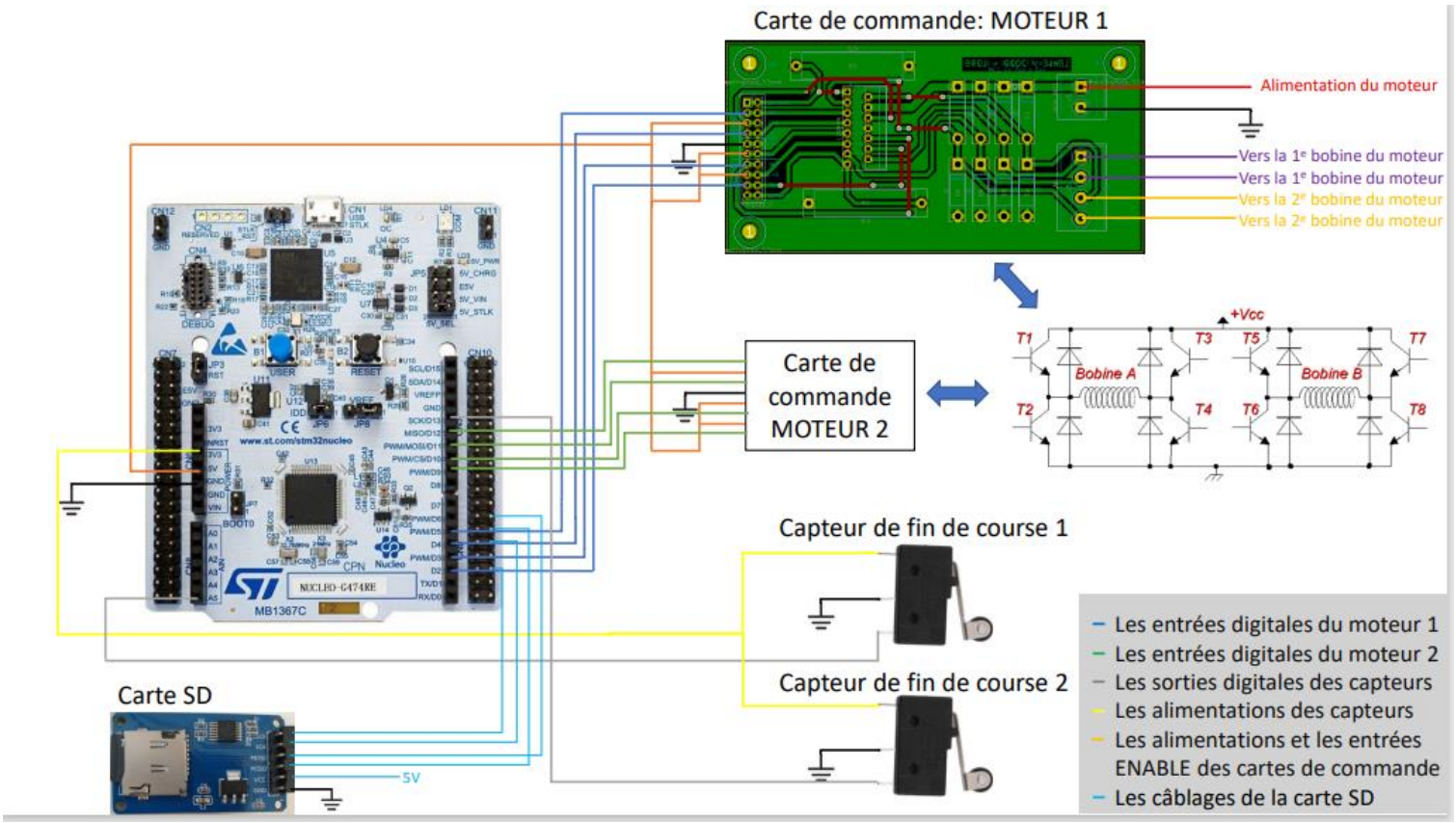


Figure 2 : Schéma du montage complet

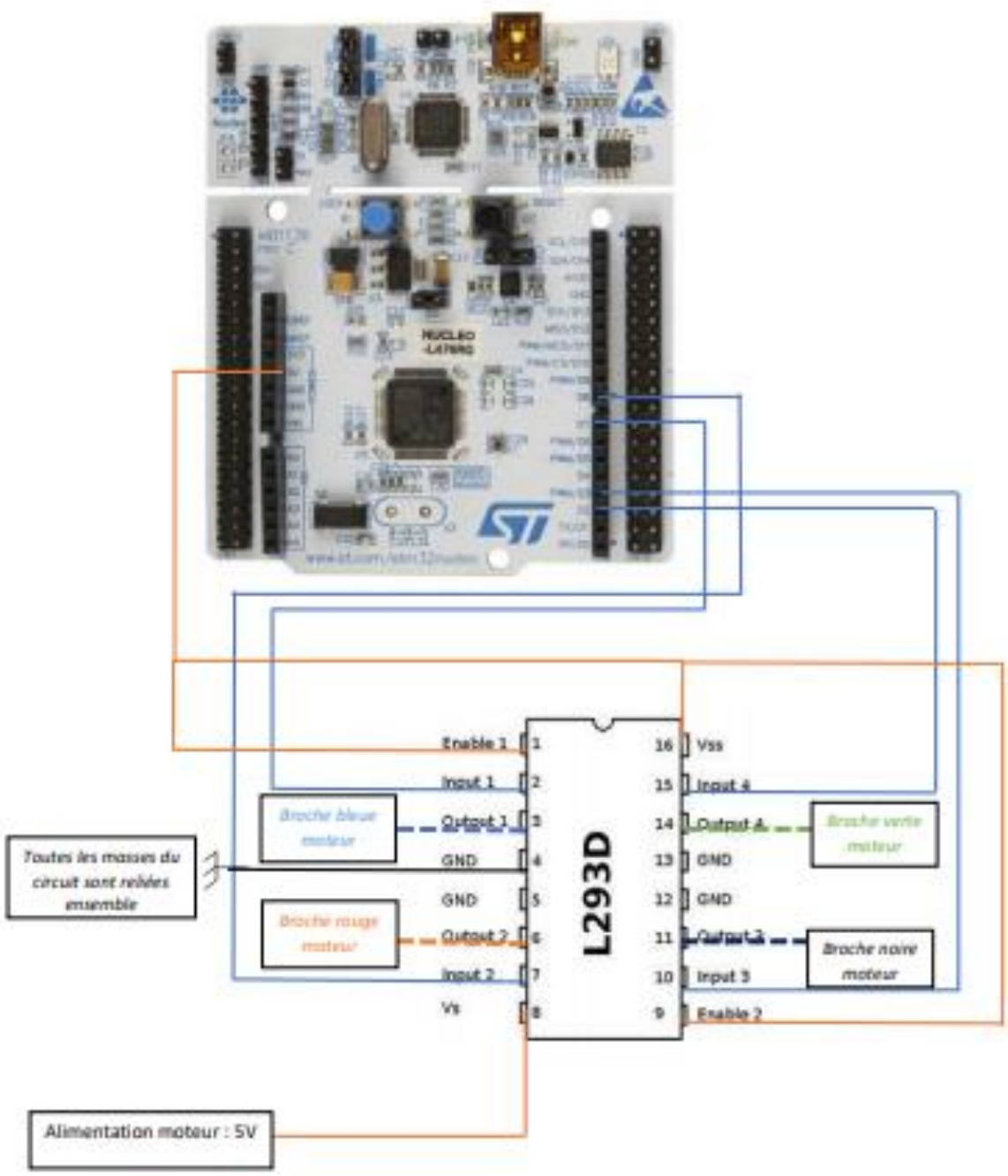


Figure 3 : Exemple de câblages pour un moteur pas à pas isolé

Algorithmes

Les algorithmes présentés dans cette partie ont été écrits en C[1], et son compilables sous Mbed pour une utilisation par un microcontrôleur comme la carte Nucléo de type L476RG[2]. Ils sont composés de trois parties : la création et l'initialisation des variables globales ; la fonction main() correspondant au programme ; l'ensemble des fonctions utilisées par le programme. Une utilisation modulaire par la création d'une bibliothèque de fonction a été réfléchie pour simplifier la lecture, sans que l'idée ne soit réalisée.
Dans la première partie nous allons présenter les variables et les fonctions permettant la réalisation d'une trajectoire complexe, la seconde partie présentera les différentes stratégies utilisées pour générer ces trajectoires, enfin nous donnerons un aperçu global du programme par la description de la fonction *main()*.

I – Fonctions générales

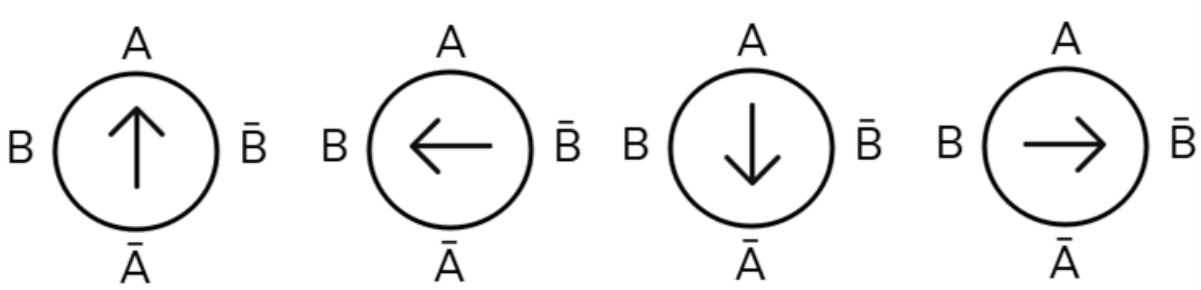
Les fonctions générales correspondent à l'ensemble des méthodes gérant le déplacement des moteurs en opposition aux fonctions de la seconde partie spécifiques à la méthode d'obtention des listes de déplacement. La première partie est consacrée au pilotage d'un moteur, la seconde au pilotage d'une trajectoire par le stylo. La dernière partie présente la méthode d'initialisation de la table et positionnement du stylo.

I-A. Rotation des moteurs pas à pas

Les deux moteurs ont un fonctionnement identique, ainsi l'initialisation des variables et les entrées et sorties de la carte est réalisé symétriquement pour chaque moteur. Nous présenterons le fonctionnement d'un seul moteur. La rotation du moteur est réalisable par un pont H piloté par le microcontrôleur. On déclare quatre sorties numériques permettant de contrôler chaque pas du moteur. La vitesse de rotation est contrôlée par la période d'interruption de la fonction réalisant le déplacement.
Un schéma de principe est donné *Figure 1*. Il présente le principe d'un moteur à un pas. Les sorties numériques sont successivement mises à jour pour réaliser la rotation.

[1] Le programme et sa version compilée sont disponibles à l'adresse suivante : https://owncloud.institutoptique.fr/s/ngfHf2epE8qpp3j Dans la suite, le nom des fonctions seront placés [entre crochet]

[2] Une liaison avec Teraterm est possible pour suivre les étapes réalisées par la carte Nucléo, l'envoi des informations n'ont pas été ajoutés aux algorithmes pour simplifier leur lecture. Rappelons que l'affichage ralenti le fonctionnement du microcontrôleur.



Étape	Valeurs des sorties numériques			
	A	B	Ā	B̄
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

Tableau 1 – Schéma de l'avancement d'un moteur pas à pas pour une rotation dans le sens trigonométrique

Nous avons écrit deux fonctions, car les étapes sont réalisées dans un sens différent pour une rotation dans le sens horaire ou anti-horaire. Nous utilisons des structures conditionnelles, l'état à l'entrée détermine l'état en sortie grâce au tableau de la Figure 1 retranscrit dans l'a*lgorithme 1.*

```
Si Entrée (A, B, Ā, B̄)= (1,0,0,0) alors Sortie = (0,1,0,0)
Sinon
        Si Entrée = (0,1,0,0) alors Sortie = (0,0,1,0)
        Sinon
                Si Entrée = (0,0,1,0) alors Sortie = (0,0,0,1)
                Sinon Sortie = (1,0,0,0)
```

Algorithme 1 – Rotation d'un pas de moteur dans le sens trigonométrique

Chaque mouvement est comptabilisé dans une variable qui s'incrémente d'une unité à chaque modification des sorties numériques. Cette variable augmente pour une rotation dans le sens horaire et diminue d'une unité pour une rotation inverse.
Enfin les quatre fonctions de rotation des moteurs sont des fonctions d'interruption de la boucle principale indépendantes. Cela nécessite un contrôle de la réalisation des rotations par une variable binaire. La rotation dans le sens horaire est réalisée uniquement si cette variable vaut 1, si elle vaut -1 le moteur tournera dans le sens trigonométrique sinon il ne se passe rien.[3]
Finalement la fonction réalisant une rotation d'un pas en sens trigonométrique peut être résumée par l'*algorithme 2.*

[3]Cette variable est dédoublée lorsque le sens de rotation est contrôlé par des boutons poussoirs (pavé directionnel) : chaque bouton contrôle un sens de rotation, la vitesse des moteurs reste constante *(voir I.B)*

Si *sens_rotation* = −1
compteur = *compteur* – *1*
Algorithme 1

Algorithme 2 – Fonction de rotation dans le sens trigonométrique *[toggle_interruptor_trigom1 – toggle_interruptor_trigom2]*

Ainsi, en assignant la valeur zéro à la variable binaire « sens_rotation », on bloque la rotation du moteur. C'est le rôle de la fonction *stop_moteurs().*

I.B – Réalisation d'une trajectoire

Une trajectoire est complétement décrite par le nombre de pas à réaliser par chacun des moteurs. Ces nombres sont des entiers relatifs, le signe détermine le sens de rotation du moteur. L'exemple de la *Figure 2,* permet de clarifier notre paramétrage. Une rotation du moteur 2 dans le sens horaire du moteur 2 réalisera un déplacement vers à la droite et une rotation trigonométrique déplace vers la gauche (le compteur diminue *voir I.A).* Le même formalisme est appliqué au moteur 1.

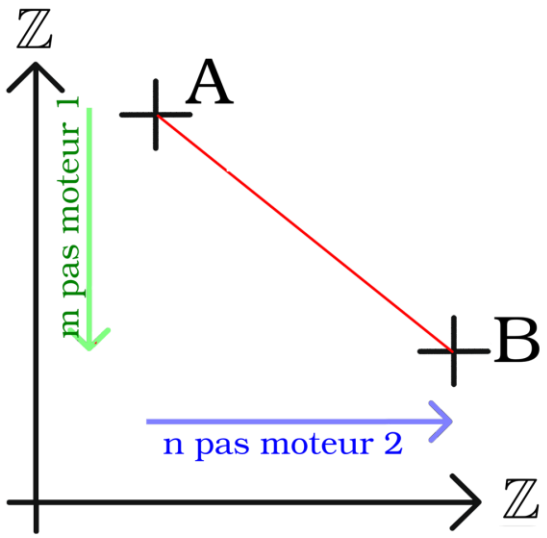


Figure 4 – Schéma d'une trajectoire simple d'un point A à un point B (n> 0 et m <0)

<u>Mise à jour de la vitesse des moteurs</u>

Les deux moteurs terminent respectivement leur m et n pas au même instant. Pour ce faire, on met à jour la vitesse des moteurs en modifiant la période d'interruption des fonctions de déplacement. L'*algorithme 3* présente la solution choisie.

```
Entrée :
        nombre pas moteur 1 : m
        nombre pas moteur 2 : n
Début :
        Si m = 0 :                      % aucun déplacement du moteur 1
                perInterM1= ∞           % période interruption de la fonction déplacement moteur 1
        Si n = 0 :
                perInterM2 = ∞
        Sinon
                rap = |m /n|            % le rapport est strictement positif
                perInterM2 = perInterM2 * rap
        Mise à jour de la période interruption des fonctions déplacements
Fin
```

Algorithme 3 – Calcul et mise à jour de la vitesse des moteurs *[maj_vitesse_moteurs]*

Pour faciliter la compréhension, on peut considérer les cas simples où m= n puis m = n/2, pour n pair. Dans le premier cas, rap vaut 1 et les deux vitesses des moteurs sont égales. On parcourt la diagonale d'un carré. Dans le second cas, le rapport vaut 1/2. Le moteur 2 se déplace deux fois moins souvent que le moteur 1, on dira alors que sa vitesse a été divisée d'un facteur 2.
Cette solution est limitée lorsque l'écart entre les deux vitesses et très important ou très petit en particulier pour de petit déplacements. De plus le moteur 2 se retrouve parfois à tourner vite ou lentement tandis que le moteur 1 se déplace toujours à vitesse constante : une solution positionnant systématiquement la vitesse des moteurs à des valeurs raisonnables ne préservant le rapport est envisageable.

La fonction *modif_per_ticker* permet de mettre à jour la période des fonctions de déplacements de moteurs. Notons que ces périodes sont définies pour chaque moteur, aussi elles correspondent à la période d'interruption des fonctions de rotation horaire et anti-horaire (*voir IA*).

Réalisation d'une trajectoire simple

Au départ de la trajectoire, les compteurs de mouvement du moteur sont initialisés à zéro, les moteurs sont à l'arrêt et le rapport des périodes des fonctions d'interruption des moteurs est actualisé.
On décompose le problème en trois cas : fonctionnement d'aucun moteur (n=m=0), c'est un cas dégénéré considéré pour palier d'éventuels dysfonctionnements dans l'initialisation des trajectoires ; fonctionnement d'un seul moteur et fonctionnement des deux moteurs en simultanée.

Dans chacun des deux derniers cas, on initialise la variable « sens_rotation » de chacun des moteurs en lui attribuant respectivement le signe du nombre de pas des moteurs. Dans l'exemple de la *Figure 2* on a affecté la valeur −1 au sens de rotation du moteur 1 et +1 pour celui du moteur 2. Puis on reste dans une boucle while tant que le nombre de mouvement des moteurs est inférieur ou égale au nombre de pas demandé. Les conditions utilisées sont spécifiées dans l'*algorithme 4.*
Lorsque l'on sort de la boucle tant que, on bloque les moteurs puis on réinitialise les compteurs à zéro et les périodes d'interruptions à la valeur par défaut (via la fonction *initialisation_moteur()*).

```
Entrée :
        nombre pas moteur 1 : m
        nombre pas moteur 2 : n
Début :
        Si m = 0 et n = 0 :                                          % aucun déplacement
                Réinitialiser les périodes d'interruption des Tickers
        Sinon
                sens_rotation_m1 = signe (m)
                sens_rotation_m2 = signe(n)

                tant que |compteur_m1| ≤ |m| xor |compteur_m2| ≤ |n|   % réalisée si m ou n =0
                        fin tant que
                tant que |compteur_m1| ≤ |m| et |compteur_m2| ≤ |n|    % réalisée si m et n ≠ 0
                        fin tant que

                Arrêt des moteurs
                Réinitialiser les périodes d'interruption des Tickers
Fin
```

Algorithme 4 – Réalisation d'une trajectoire simple *[deplacement]*

Réalisation d'une trajectoire complexe

Une trajectoire complexe se décompose en plusieurs trajectoires simples. Ainsi on regroupe dans des listes les trajectoires successives parcourues par les moteurs puis on parcourt chaque élément de ces listes (*Algorithme 5*).
Avant et après chaque trajectoire complexe, il est souvent nécessaire de lever ou descendre le stylo. Comme nous n'avons pas câblé de troisième moteur, nous effectuons cette action manuellement ne stoppant la rotation des moteurs tant qu'un bouton n'a pas été pressé[4] (*Algorithme 6*).

```
Entrée :
        trajectoires m1 : [m_1, m_2, m_3,…, m_dim]        % dim est le nombre de trajectoire simple
        trajectoires m2 : [n_1, n_2, n_3,…, n_dim]
Début :
        Pour k allant de 1 à dim
                Si k = 1 : descendre le stylo (Algorithme 6)

                Actualisation de la vitesse des moteurs avec m_k et n_k (Algorithme 3)
                Réalisation de la trajectoire k (Algorithme 4)

                Si k = dim ; monter le stylo (Algorithme 6)
Fin
```

Algorithme 5 – Réalisation d'une trajectoire complexe *[parcours]*

[4]La solution donnée dans le script est limitée, car nous n'avons pas câblé de bouton poussoir dédié et utilisons les boutons de fin de course utilisé dans la remise à l'origine. Il est alors impossible de réaliser les deux actions successivement.

```
Début :
        Bloquer la rotation des moteurs (I.A)
        Tant que bouton = 0                                  % bouton est une variable booléenne

                Afficher sur la console « Attente pression du bouton »    % optionnelle si le microcontrôleur
                                                                          % est relié à une console
                fin tant que
Fin
```

Algorithme 6 – Interruption de la table pour action manuelle [*stop_moteur_boutons()*]

I-C. Remise à l'origine du stylo et positionnement du stylo

La table comporte deux boutons poussoirs placé aux extrémités des deux axes. Ils permettent de définir un point origine pour le stylo. Pour initialiser la table, chaque moteur tourne dans cette direction tant que son bouton poussoir n'est pas enclenché par le stylo (*Algorithmes 7 et 8*).

```
Début
        Si bouton poussoir = 0
                sens_rotation = −1              % valeur à accordé l'emplacement de la butée
        Sinon
                sens_rotation = 0          % arrêt du moteur
Fin
```

Algorithme 7 – Mise à jour des variables de mouvement d'un moteur jusqu'à un bouton poussoirs *[remise_originem1(), remise_originem2()]*

```
Début
        Tant que non (bouton poussoir_1 = 1 et bouton poussoir2 = 1)
                Positionner à l'origine le moteur 1 (Algorithme 7)
                Positionner à l'origine le moteur 2 (Algorithme 7)
                fin tant que
        Initialisation des compteurs des moteurs
Fin
```

Algorithme 8 – Remise à l'origine du stylo *[remise_origine()]*

Il est ensuite possible de positionner le stylo à une position repérée sur la table en déplaçant le stylo depuis l'origine *(Cf Algorithme 4).* C'est le rôle joué par la fonction *centrer_stylo,* dans laquelle nous avons estimé le nombre de pas nécessaires pour placer le stylo au centre de la table.

II – Obtention des listes de trajectoires

Cette partie s'intéresse aux différentes méthodes utilisées pour obtenir les listes contenant les trajectoires. Trois méthodes ont été imaginées. Les deux premières méthodes sont détaillées, les résultats de ses deux stratégies sont consultables sur la vidéo en lien avec ce rapport. La troisième méthode ainsi que les prolongements qu'elle offre seront présentés brièvement.

II-A. Calcul d'une trajectoire par le microcontrôleur

Cette première méthode est utilisée pour tracer des courbes planes dont on connaît les équations paramétriques. On échantillonne la courbe sur un nombre de point donnée puis on calcule l'écart entre chaque point qu'on arrondit à l'entier le plus proche (*Algorithme 9*).

```
Entrée :
        nombre de points : NBPoint
        fonctions (x(t), y(t)) de la courbe
        pas de temps entre chaque point : dtx, dty
Début
        Création de listes X, Y de taille NBPoint
        X[0], Y[0] = x(0), y(0)                          % initialisation des listes d'échantillons
        Pour k allant de 0 à NBPoint-1
                X[k+1] = x((dtx+1)*k)                    % échantillonnage de la courbe
                Y[k+1] = y((dty+1)*k)

                trajectoire_m1 = round(Y[k+1]-Y[k])      % conversion en nombre de pas relatif
                trajectoire_m2 = round(X[k+1]-X[k])
Fin
```

Algorithme 9 – Méthode générale pour dessiner une courbe mathématique

Nous avons appliqué cette stratégie aux cas particuliers des courbes fermée pour tracer un cercle ou des cycloïdes *[epicycloide(), hypocycloide()]*. L'*algorithme 10* présente plus précisément la génération des listes d'une épicycloïde. Rappelons que ces courbes sont obtenues en considérant un point d'un cercle roulant sur un cercle directeur de rayon R. La courbe est alors complétement définie par la donnée du rayon du cercle roulant r et le rapport des rayons q égal à r/R selon les équations paramétriques suivantes, pour θ allant de 0 à 2π :

$$x(\theta) = r((q+1) \times \cos(\theta) - \cos\big((q+1) \times \theta\big))$$
$$y(\theta) = r((q+1) \times \sin(\theta) - \sin\left((q+1) \times \theta\right)$$

```
Entrée :
        nombre de points : NBPoints
        paramètres de l'épicycloïde : r et q
Début
        Création de listes X, Y de taille NBPoints contenant des réels
        θ = 0
        X[0] = r[(q+1) cos(θ) – cos ((q+1)θ)]
        Y[0] = r[(q+1) sin(θ) – sin ((q+1)θ)]

        Pour k allant de 0 à NBPoints-1              % échantillonnage de la courbe
                θ = 2π * (k+1)/ NBPoints
                X[k+1] = r[(q+1) cos(θ) – cos ((q+1)θ)]
                Y[k+1] = r[(q+1) sin(θ) – sin ((q+1)θ)]

                trajectoire_m1 = round(Y[k+1]-Y[k])      % conversion en nombre de pas relatif
                trajectoire_m2 = round(X[k+1]-X[k])
Fin
```

Algorithme 10 – Génération des listes de trajectoire d'une épicycloïde *[epycloide()]*

II-B. Implantation d'une trajectoire sur le microcontrôleur

La solution de l'*algorithme 9* révèle ses limites lorsqu'on considère un grand nombre de point ou des fonctions à grandes complexités[5]. La compilation du programme est alors impossible. Ceci nous a amené à échantillonner au préalable les figures dans une console séparée. Les listes de réel sont copiées dans une fonction qui remplit les listes de trajectoire des moteurs 1 et 2 en arrondissant à l'entier le plus proche la différence respectivement d'ordonnée et d'abscisse (*Algorithme 11*).
Cette méthode a été utilisée pour réaliser la figure du Mickey Mouse par une interpolation trigonométrique de 32 harmoniques sur 1000 points.

```
Début
        listes de X, Y comportant NBPoints points préalablement calculer
        Pour k allant de 0 à NBPoints – 1
                trajectoire_m1 = round(Y[k+1]-Y[k])      % conversion en nombre de pas relatif
                trajectoire_m2 = round(X[k+1]-X[k])
Fin
```

Algorithme 11 – Conversion d'un ensemble de point d'une trajectoire précise préalablement calculé en déplacement relatif pour les moteurs pas à pas [*mickey()]*

II-C. Lecture de trajectoire stockée sur un espace mémoire par le microcontrôleur et prolongements (à titre indicatif)

La stratégie précédente peut être développée en stockant les trajectoires dans un espace mémoire. On peut alors généraliser l'*algorithme 11* en donnant les listes X et Y comme variable d'entrée d'une fonction de conversion en déplacement relatif des moteurs. On peut de surcroît directement stocker les trajectoires converties en déplacement de moteur. Il convient alors de mettre en place une procédure permettant de l'obtention de ces trajectoires.
Les trajectoires sont stockées dans des fichiers texte. Ce fichier comporte sur la première le nombre de trajectoire, puis les informations relatives à chaque trajectoire sur trois lignes : le nombre de points, les déplacements successifs du moteur 1 et ceux du moteur 2. Les trajectoires sont alors initialisées et réalisées selon l'*algorithme 12.*

```
Si ouverture du fichier réussie :
        lire le nombre de trace et initialiser l'entier nb_trace
        Pour k allant de 1 à  nb_trace
                lire les trajectoires k du fichier [lecture_trajet()]
                réaliser la trajectoire (Algorithme 5)
        fermer le fichier
```

Algorithme 12 – Lecture d'un fichier comportant plusieurs trajectoires et réalisation de ces trajectoires *[mouvement()]*

Cette méthode de récupération de l'information nous rend complétement indépendant des limites de calcul du compilateur. Nous avons par ailleurs développé deux procédés pour générer des fichiers. L'un à partir d'une image SVG comportant des courbes de Béziers

[5] l'*algorithme 10* d'algorithme peut être élargi à toutes courbes fermées quelconque en remplaçant les paramètres d'entrée par les polynômes trigonométriques interpolateurs de la courbe.

composées uniquement de segment de droite, le second est basé sur une interface spécifiquement codée offrant la possibilité de placer ses propres points (voir *Figure 2*) notamment pour relever les contours d'une image.[6]

III – Programme principal

Le programme principal permettant de dessiner la figure souhaitée correspond à la fonction *main().* Il initialise la position du stylo, puis calcul ou récupère les trajectoires, réalise les mouvements et retourne se placer à l'origine. (*Algorithme 13*)

```
Début
        Remise à l'origine (Algorithme 8)
        Placement du stylo (Algorithme 4)

        Initialisation des trajectoires
        Pour k allant de 1 à nombre_de_trajectoire :
                Initialisation de la trajectoire k
                Réalisation de la trajectoire k (Algorithme 5)

        Remise à l'origine (Algorithme 8)
Fin
```

Algorithme 13 – Programme optimale réalisant un dessin composé de plusieurs trajectoires *[main()]*

Souvent la fonction main() n'est pas identique à celle présentée par l'*algorithme 13,* car lorsqu'une seule trajectoire est effectuée, la boucle for est superfétatoire. Le tracé de multiple trajectoire reste possible avec les méthodes de génération de liste (IIA) et (II.B) mais la méthode (II.C) telle que présentée réalise dans la même fonction *mouvement()* l'initialisation des trajectoires et l'itération associée. Le schéma-bloc "action du microcontrôleur" résume les informations présentées dans cette partie.

Tests et validations

Les tests des fonctions et des conditions ont été réalisés progressivement lors de leur conception. En particulier les conditions des boucles while peuvent mener à des boucles infinies. Ces tests ont été réalisé indépendamment de la table via une liaison de type RS232 entre le microcontrôleur et le logiciel Teraterm sur l'ordinateur. On peut alors suivre la mise à jour des variables en jalonnant le programme de commande d'affichage. Certaines de ces commandes sont consultables sur le script en annexe. Nous présentons dans la suite de cette partie les résultats finaux avec les dessins réalisés.

[6]Des informations complémentaires ainsi que les scripts sont disponibles à l'adresse suivante : https://owncloud.institutoptique.fr/s/ngfHf2epE8qpp3j

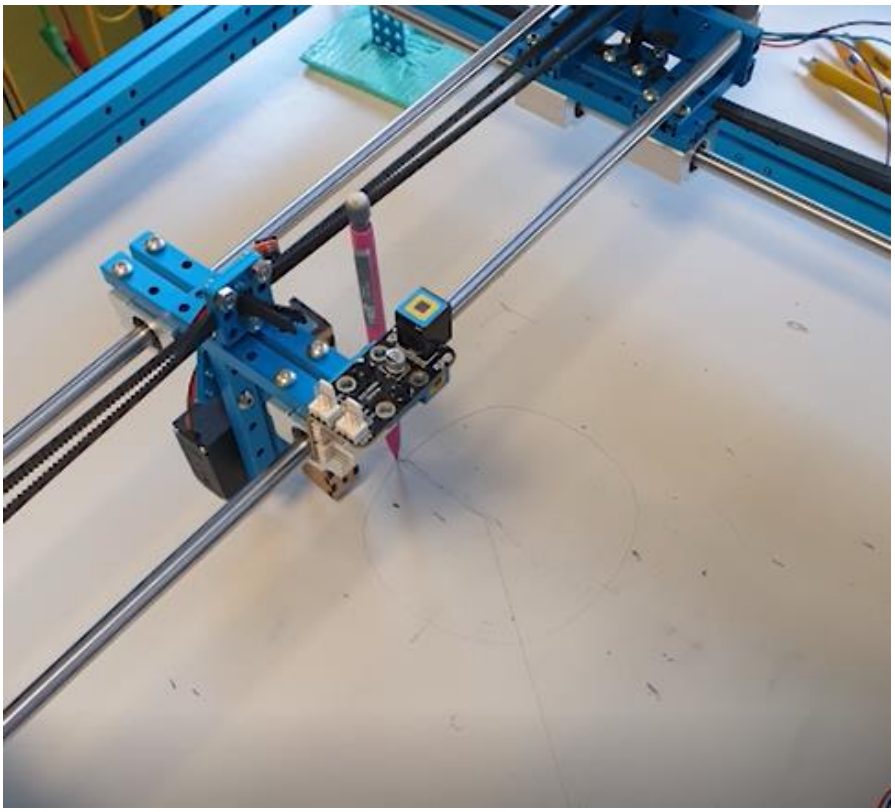


Figure 5 : Tracé du cercle

Le tracé du cercle est assez régulier pour une valeur optimum de 500 points et une vitesse faible des moteurs. Si le nombre de points ou la vitesse augmente, nous avons obtenue des hexagones. Nous n'avons pas poussé les investigations et sommes restées dans les plages optimales.

Figure 6 : Epicycloïde à trois branches

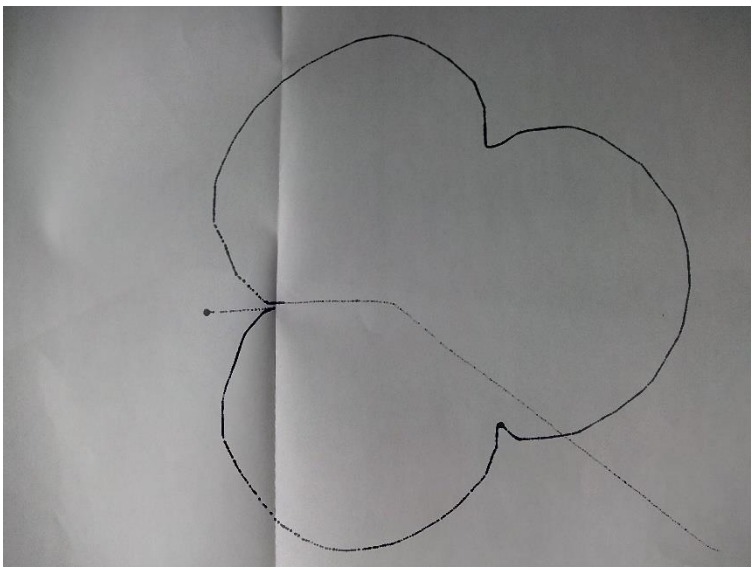


Figure 7 : Tracé de l'épicycloïde à trois branches

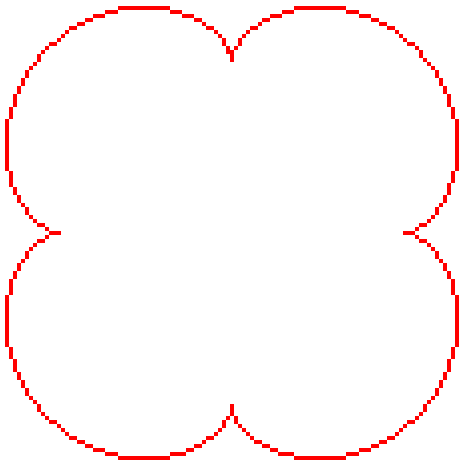


Figure 8: Epicycloïde à quatre branches

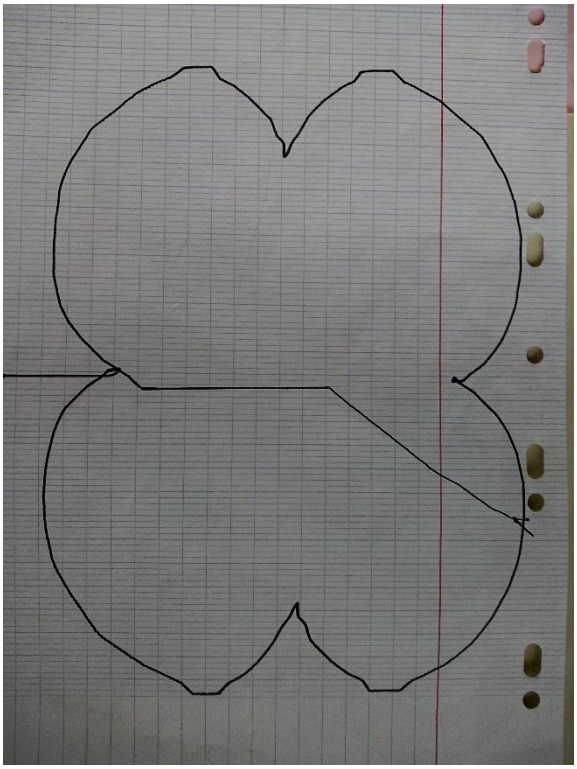


Figure 9 : Tracer de l'épicycloïde à quatre branches

L'épicycloïde a trois branches est plus régulière que celle à quatre branches. Les traits parasites correspondent au positionnement initial du stylo.

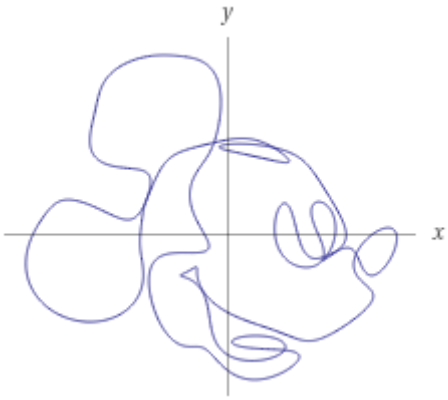


Figure 10 : Dessin de Mickey Mouse

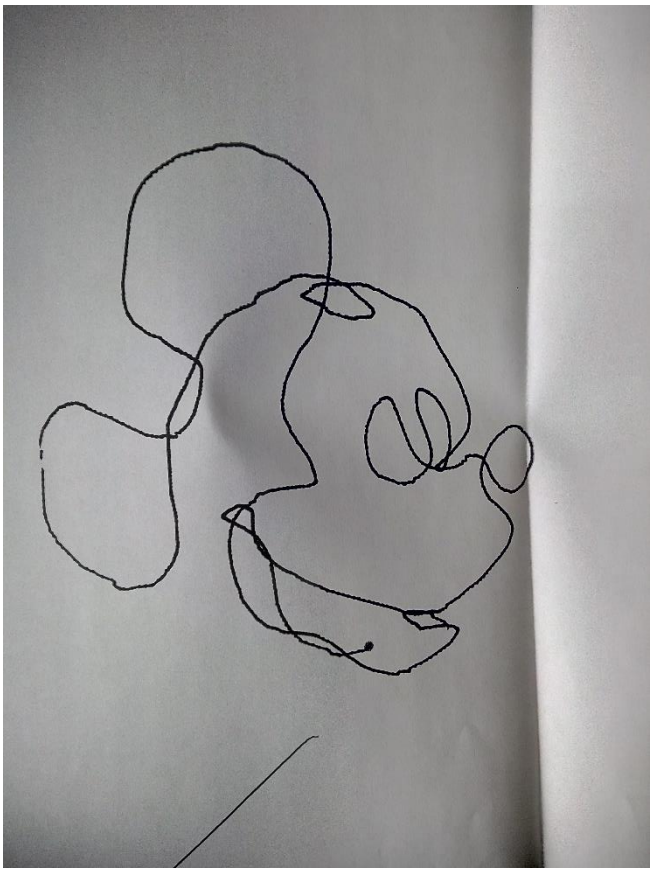


Figure 11 : Tracé du dessin de Mickey Mouse à partir d'équations paramétriques

Le Mickey est assez précis, mais la figure est plus tassée. Cela s'explique d'une part par les approximations et les imprécisions constatées précédemment et d'autre par les frottements du stylo qui entraîne un décalage régulier.

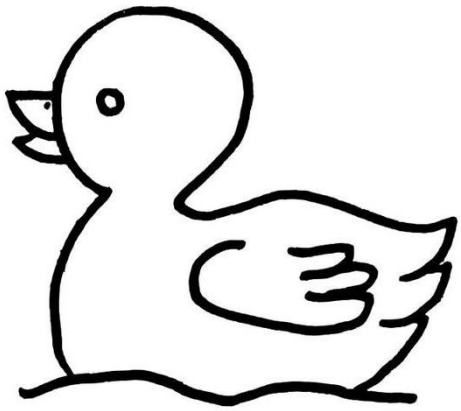


Figure 12 : Image d'origine du canard

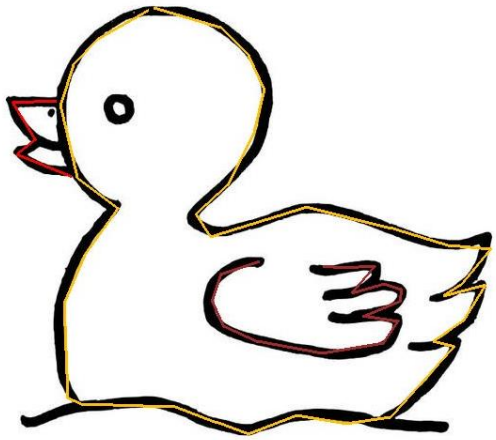


Figure 13 : Tracé à la main des contours du canard

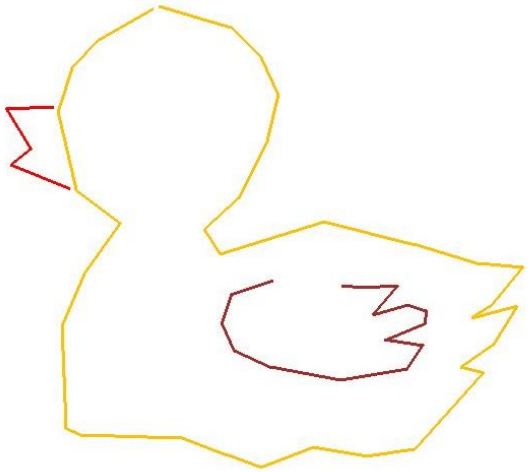


Figure 14 : Image du canard allant être tracée par la table

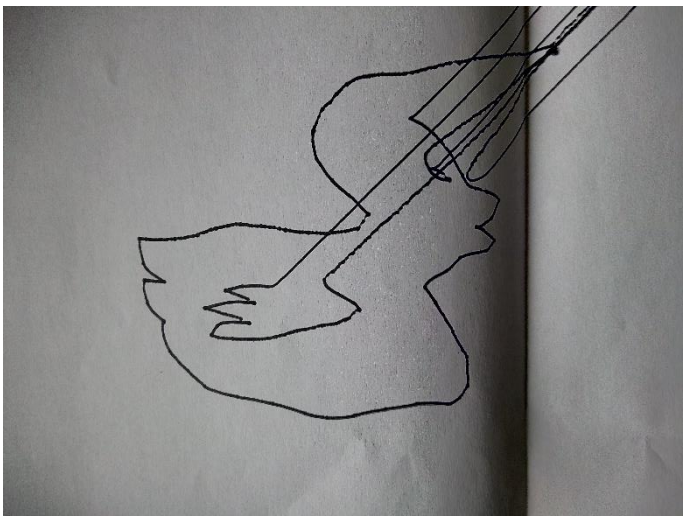


Figure 15 Tracé du canard

Le canard est réalisé de façon assez précise par cette méthode. La figure 15 présente un des premiers résultats obtenus par cette méthode. Les traits parallèles correspondent au passage d'une trace à l'autre réalisé par un retour à l'origine systématique. Ces déplacements parasites ont été par suite été évité et un protocole pour lever du stylo a été codé.

Planning

❖ Séance 1 :

Baptiste / Xinxin :

-Branchements complets moteur et carte nucléo pour le moteur isolé puis sur les moteurs de la table traçante
-Branchements moteur/cartes
-Soudure jonction

Étienne / Thadek :

-Programmation moteur pas à pas sur Mbed

-Moteur pas à pas tournant en continu

-Pilotage de la rotation du moteur en appuyant sur un bouton

-Commande du sens de rotation du moteur avec deux boutons

❖ Séance 2 :

Etienne / Thadek :

-Programmation pour utiliser les deux moteurs simultanément

-Ecriture d'un programme permettant d'utiliser quatre boutons poussoirs pour piloter le stylo dans les quatre directions et les diagonales (45°)

Baptiste / Xinxin :

- Branchements complets des 4 boutons poussoirs

-Soudure de la seconde jonction

-Élaboration d'un planning et de schémas blocs

❖ Séance 3 :

Étienne / Baptiste :

- Écriture d'un programme pour gérer la rotation des moteurs individuellement et simultanément à vitesse constante sans action extérieur : automatisation du tracé d'un carré et l'une de ses diagonales

Xinxin / Thadek :

-Câblage des interrupteurs de fin de course

-Ecriture d'un programme de remise à l'origine

❖ Séance 4 :

Étienne / Baptiste :

- Fin de programmation du tracé des diagonales à 45° et test de toutes les possibilités offertes à ce stade par notre code : tracés de triangles par exemple ou d'autres dessins plus complexes selon le choix des listes de points et leur longueur (tout en étant limités par nos contraintes sur les listes de points dues à la vitesse des moteurs qui doit être identique s'ils fonctionnement simultanément)
- Début de programmation (complexe) du tracé de droites pour des inclinaisons différentes de 45° par rapport aux axes de la table (soit des moteurs allant à des vitesses distinctes et pilotables)

Xinxin /Thadek :

- Ecriture d'un programme permettant de tracer des diagonales de pentes différentes avec une deuxième méthode. Méthode abandonnée car ne fonctionnait pas
- Mesure des paramètres de la table et réflexions sur l'échantillonnage d'unes courbes

❖ Séance 5 :

Étienne / Baptiste :

- Programmation afin de pouvoir tracer une figure géométrique segmentée quelconque à partir de deux listes de points (une pour chaque moteur) : c'est donc la finalisation de ce bloc avec cette possibilité de tracer des diagonales pour des inclinaisons différentes de 45°
- Recherche de solutions pour implémenter efficacement les listes de points à partir de courbes ou dessins choisis afin d'être tracés

Xinxin / Thadek :

- Déplacement et câblage de la table suite à un ordinateur qui tombe en panne et une carte nucléo qui crame

- Écriture d'un programme permettant de tracer un cercle, le résultat est un octogone ou un cercle plus ou moins régulier selon le nombre de points utilisés pour discrétiser

❖ Séance 6 :

Xinxin / Thadek :

- Obtention d'un cercle régulier à partir du programme de la séance 5
- Tracés des épicycloïdes de rapport différents (notamment 3 et 4)
- Tester et filmer pour les livrables

Baptiste / Étienne :

- Amélioration et clarification des programmes du microcontrôleur
- Recherches pour réaliser un dessin sur la table à partir d'une image SVG (Script Python)
- Débogage des premiers tests

❖ Séance 7 :

Xinxin / Thadek :

- Amélioration du programme de traçage et tracé du Mickey à partir de son équation.
- Filmer les derniers programmes restants pour les livrables

Étienne / Baptiste :

- Débogage des nouveaux programmes
- Câblage d'un lecteur de carte SD et mise à jour de code de lecture de fichier pour une utilisation sur Mbed
- Premier test de l'interface Pygame

❖ Séance 8 :

- Branchements et derniers tests de fonctionnement de la table (vérification)
- Présentation finale du projet

Difficultés rencontrées et analyse du travail d'équipe

Ce projet a rencontré assez peu de difficultés si ce n'est quelques problèmes de matériel et de montages, à cause d'une mauvaise coordination avec les autres groupes utilisant la même table traçante dont les branchements pouvaient différer des nôtres. La majeure partie des difficultés a été en lien avec l'environnement de codage Mbed : en particulier la mise à jour des bibliothèques entre les différentes versions du compilateur.

Le travail a été réparti en deux groupes de deux ce qui a permis d'être plutôt efficace. Sur les premières séances Étienne et Thadek ont réalisé les premiers programmes pour faire tourner les moteurs pendant que Baptiste et Xinxin s'occupaient des branchements. À partir de la troisième séance les groupes ont changé. Étienne et Baptiste se sont concentrés sur une méthode pour tracer des traits d'inclinaisons différentes, puis sur le tracé à partir d'un dessin à la main sur l'ordinateur. Xinxin et Thadek ont réalisé un programme de remise à l'origine, puis ont réalisé des dessins à partir d'équations paramétriques. Cependant, même si le travail était réparti, de nombreuses interactions entre les deux groupes ont permis de débloquer de nombreux problèmes. Pour conclure, la séparation en deux groupes qui réfléchissent sur des méthodes différentes a permis d'avancer assez rapidement.