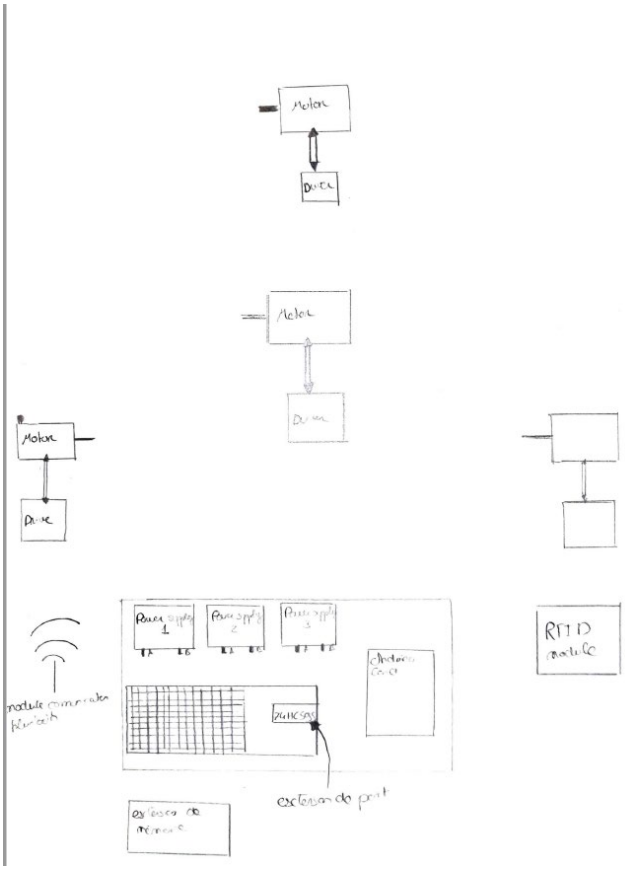
Rapport de séance semaine du 10-12-2018

Cette séance j'ai commencé à rédiger le cahier des charges ; notamment sur les objectifs de ce projet et les attentes. J'ai aussi réfléchit sur l'électronique embarqué de l'imprimante, en pensant aux interactions entre les divers composants nécessaires aux projets.

Au niveau du cahier des charges, il n'est pas nécessaire d'en dire plus étant donné qu'il est dans le GitHub.

En revanche sur l'électronique, des précisions peuvent être apportées

Voici un premier schéma très simplifier des différents composants nécessaires.



- On a besoin de 3 générateurs de courants pour alimenter les moteurs et la cartes Arduino.
- Un modem Bluetooth sera nécessaire pour recevoir les informations provenant de l'ordinateur.
- Une extension de mémoire afin d'enregistrer plusieurs pages à imprimer (qui sera sous forme de coordonnées).

- Un RTID afin de sécuriser notre imprimante.
- Un écran LCD afin de visualiser les tâches en cours, ou accéder au panneau de contrôle.
- Un pavé numérique permettant d'interagir avec la carte Arduino.
- Les drivers pour les moteurs.
- Une extension de port permettant de pouvoir tout relié à la carte arduino.

Rapport de séance semaine du 17-12-2018

Cette semaine je me suis concentré sur une seule chose, réussir à importer les données de la base de l'EMNIST dans deux tableau python.

Cette étape semble facile, mais ce n'est pas du tout le cas. Il y a très peu de documentation sur l'EMNIST car la plupart des exemples sont fait avec l'MNIST (une version bien moins complète).

La base de l'EMNIST est une ressource divisée en plusieurs fichiers et qui existe en plusieurs versions, d'un poids de 700 mo, elle ne comporte pas moins de 700 000 caractères (chiffres et lettres).

On le charge sous forme de deux tableau à 3 dimensions :

Le premier contient l'index du caractère et les pixels.

Le deuxième contient le caractère écrit en version ordinateur.

Au bout d'environ 3 heures, j'ai réussi à mixer plusieurs codes pour pouvoir charger les données (il faut environ 1 minutes à un ordinateur bien au-dessus de la moyenne pour pouvoir les charger (exécution en single core), et nécessite au minimum 5.5 go de mémoire vive non utilisé).

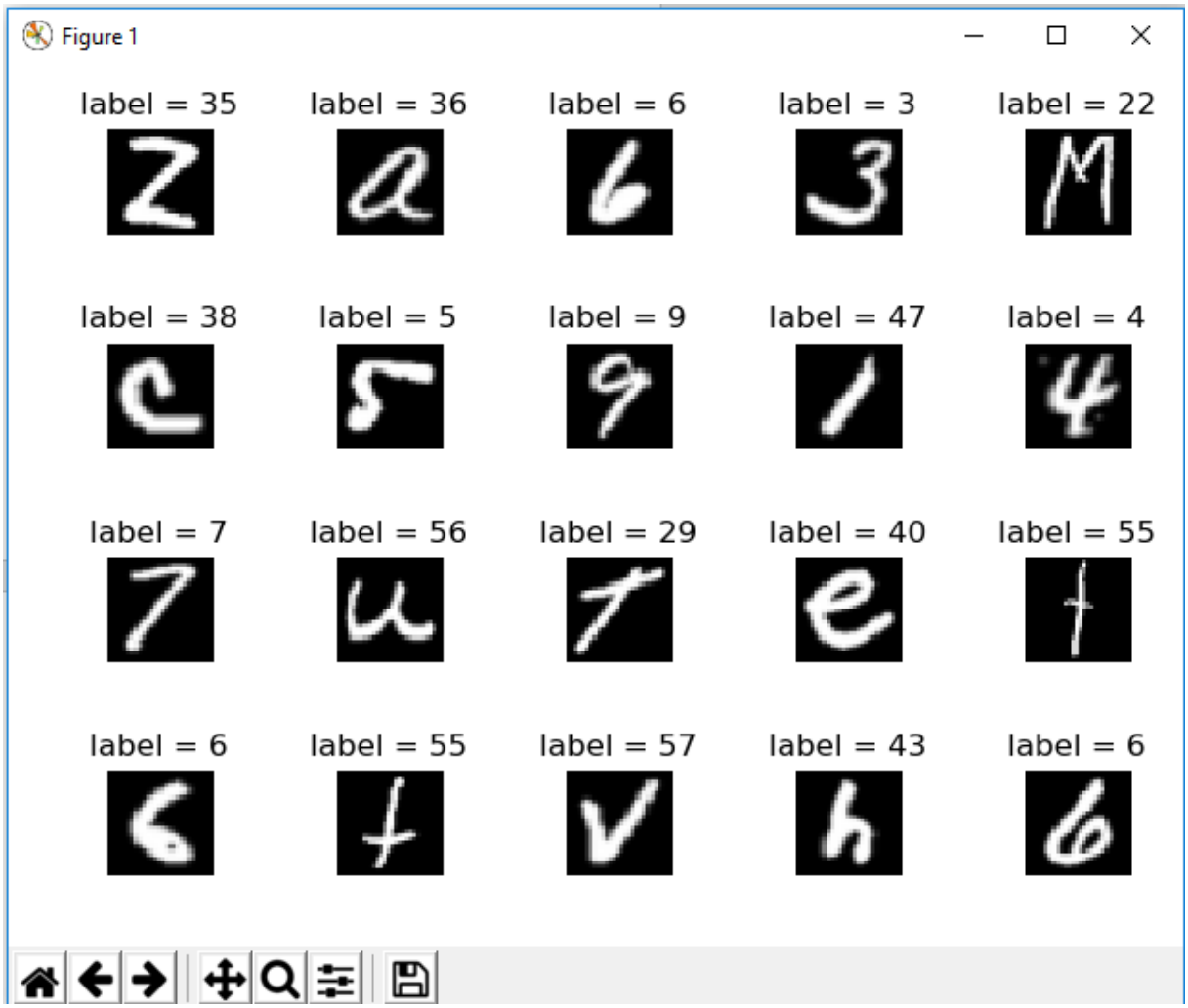
Le code est au final minuscule car il j'ai trouvé une classe (MNIST) déjà toute faite qui s'occupe de tout, sinon le code aurait était très compliqué (manipulation de gros tableaux avec beaucoup de transformations).

J'ai écrit ce tableau alliant label et correspondance

0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F
16	G
17	H
18	I
19	J
20	K
21	L
22	M
23	N
24	O
25	P
26	Q
27	R
28	S
29	T

30	U
31	V
32	W
33	X
34	Y
35	Z
36	a
37	b
38	c
39	d
40	e
41	f
42	g
43	h
45	i
46	j
47	k
48	l
49	m
50	o
51	p
52	k
53	r
54	s
55	t
56	u
57	v
58	w
59	z
60	y
61	z

Après avoir finalement créer le tableau, j'ai tester, et voici quelques résultats (une infime partie) :



On peut maintenant utiliser ces données pour créer le premier réseau de neurones capable de reconnaitre les caractère .

Information : Le fichier de données est trop lourd pour être portée sur GitHub

Rapport vacance de décembre-07/01/2019 Antoine

Vacances :

J'ai décidé pendant les vacances de décembre de développer la partie code de notre projet. Je devais donc être capable de reproduire sur une image numérique des phrases manuscrites identiques à celles de Raphael mais toutes générer aléatoirement.

La première étape fût donc de récupérer des pages d'écritures. On a commencé par 80 pages de cours de maths.

Initialement nous voulions utiliser la base de donnée IAM afin de reconnaitre des mots et de les analyser, seulement on n'avait pas pensé que la langue dans laquelle était la base de donnée allait nous l'empêcher. En effet en utilisant IAM on aurait pu lire directement le mot sans lire les lettres une par une (c'est ce que fait un humain) cependant on aurait pu lire des mots anglais mais Raphael n'est pas anglais

Nota : La base de l'IAM est une base donnée regroupant 115 000 mots, pouvant être séparer en lettres. En fait on dispose IAM d'une grande quantité de fichier XML regroupant des lignes de textes, séparés en mots puis en lettre. On dispose aussi d'une multitude de suite de points (coordonnées) qui sont des séquences pour écrire de la lettre a lettre

Lien : *http://www.fki.inf.unibe.ch/databases/iam-handwriting-database*

Il a donc fallu trouver une autre méthode

J'ai donc créer un algorithme séparant chaque mot de l'image, afin de pouvoir travailler mot à mot. Cela n'a pas été extrêmement difficile, en une journée c'était fait.

Un gros problème (*non résolu*) fût de splitter chaque lettre du mot en petites images, évidement lorsque la personne n'écrit pas en cursive c'est facile, mais Raphael fait un mélange des deux et n'a pas une écriture stable et bien définie.

J'ai donc tenter d'analyser la densité de pixel :

Cette courbe représente la densité de pixel blanc par rapport à la position horizontal dans l'image. Comme vous pouvez le voir on peut rien en déduire.

L'analyse du gradient de cette courbe ne mène aussi à rien

Deuxième technique : détection de formes :

(Ne mène aussi à rien)

DONC :

Les diverses techniques employés (même en les combinant, en calculant la moyenne de la densité de pixel etc…) ne mènent à rien.

Après avoir longuement réfléchis je me suis ne dit pourquoi pas utiliser la détection de lettre par réseaux de neurones en balayant chaque mot directement, sans faire de la lettre à lettre.

J'ai donc entrainer un réseau de neurones (code dans « train.py ») permettant de reconnaitre des chiffres, lettres minuscule et lettres majuscules. En l'entraînant environ 1h, le résultat stagné a 85-86% pourcent de bonnes réponses sur la base de données de l'Emnist.

Cependant sur l'écriture de Raphael, les réseaux de neurones se trompait dans environ 40% des cas, notamment au niveaux des « 1 » qui étaient toujours détectés comme des « A ».

Il a donc fallu « nourrir » le réseau de neurones avec un nouveau jeu d'entrainement, celui de Raphael. J'ai donc demander à Raphael de beaucoup écrire, notamment des lettres uniques, afin des créer de nouvelles données catégorisées dans des fichiers.

Après avoir accumuler et triées plus de 2500 caractères, les tests s'avèrent plus généreux avec environ 8% erreurs.

Cependant j'avais oublié quelque chose, lorsque le balayage allait se faire, même si le réseau de neurones prenait un mot en entier, il allait choisit parmi les classes prédéfinies, et donc par exemple, si on lui donne en entrée le mot « elle » il donnerai en sorti une lettre ou un chiffre, ce qui ne serait pas bon du tout ! Il a donc fallu rajouter la classe « autre », si le réseau de neurone détecté autre chose qu'un caractère, il choisissait « autre ».

J'ai donc essayé d'utiliser le balayage afin de détecter chaque lettres d'un mot, mais le résultat s'est avérer bien mauvais, et je n'avoue ne pas savoir pourquoi.

Peut-être mon réseau de neurones est trop petit, ou bien peut-être faut-il une quantité folle de données pour d'approcher d'un résultat correct

Le plan était initialement d'avoir ses mots référencés afin de les infecter dans un autre réseau de neurones pouvant apprendre à générer du texte, cependant j'ai appris qu'il fallait écrire sur un tableau interactif afin d'enregistrer une succession de points lors de l'écriture et qu'une image ne sert à rien, on est pas capable d'apprendre à écrire à une machine avec une image, mais seulement avec une série de points.

Afin de contourner l'impasse j'ai voulu utiliser le premier réseau de neurones afin d'obtenir l'espacement entre les lettres.

Par exemple combien de pixel entre un « p » et un « u », un « n » et un « e ».

Le code contenu dans « character-extrator.py » était là pour ça mais comme dit précédemment le réseau de neurones n'est pas assez performant.

Mais je me suis contenter de son résultat. Les lettres étaient à peu près bien détecter (le mot était bien découpé), mais les prédictions étaient souvent fausses.

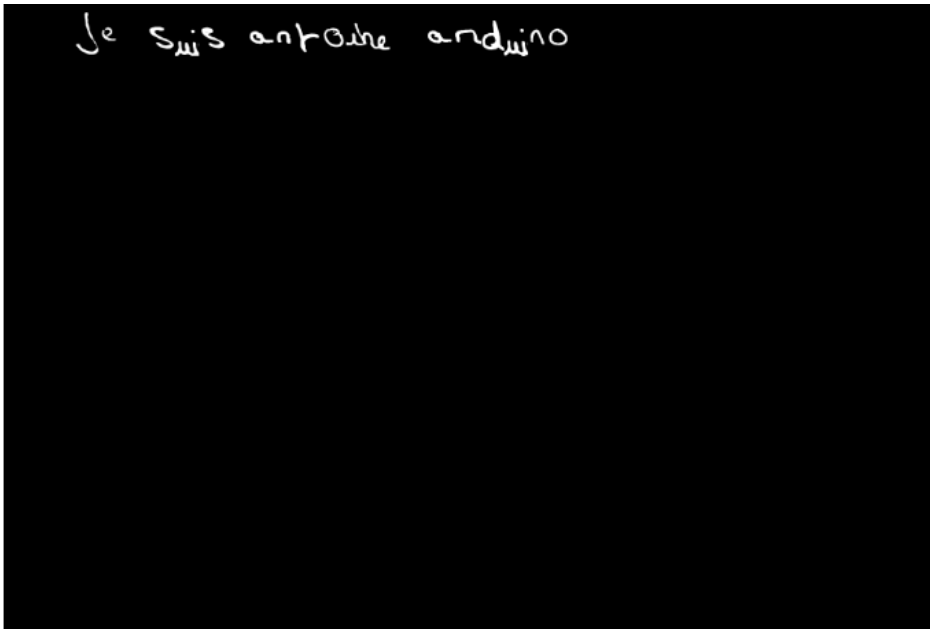
J'ai donc récupérer les mots à peu près bien découpé et ensuite corriger les erreurs de prédictions (n au lieu de r pas exemple).
Je pense que avec 10 000 images on aurait pu avoir un bon taux de prédictions car il faut savoir que les lettres ou on avait donné les plus d'images (ex : « e »), les détections étaient souvent juste.

Maintenant qu'on a nos lettres et mots triés, il nous reste à faire le code permettant de construire la page et de communiquer avec l'imprimante.

J'ai commencé ce code …

07/01/2019 :

Lors de cette séance, j'ai finaliser la version 1 du code permettant de générer des pages de textes. Sans plus attendre voici juste quelques mots écrits par mon algorithme :



Je compte améliorer les liaisons entre caractères, et augmenter considérablement le texte écrit (même si je le peux déjà mais le résultat n'est pas terrible). Cependant il m'est presque impossible de parfaitement aligner les lettres entres elles. Je ne pense pas pouvoir plus me rapprocher de l'écriture de Raphael, ce qui est fort dommage car ça ne s'en rapproche pas trop.

Il faut savoir que pour arriver à ces résultats j'utilise deux choses, des parties de mots toutes référencés et découper, grâce (en partie) a mon réseau de neurones, et plus majoritairement à ma patiente à corriger les fautes de mon réseau de neurones lors du découpage.

J'ai donc ajouter certains fichiers contenant du code, en voici la liste avec une petite description pour chacun d'entre eux :

- **Character-extractor.py** *permet à partir d'un ensemble de mots, d'extraire le plus possible de lettres, ou d'ensembles de lettres*
- **Data.py** *un petit fichier juste pour mettre quelques variables*
- **Images_utils.py** *ensemble de fonctions nécessaires aux autres modules*
- **Line.py** *fichier dans lequel se trouve la classe Line, représentant une ligne d'écriture*
- **Page.py** *fichier dans lequel se trouve la classe Page, représentant une page d'écriture*
- **Printing.py** *fichier initialiseur, lance la construction des pages, la transformation en coordonnées pour l'imprimante et la communication avec la carte arduino*
- **Train.py** *fichier permettant d'entrainer notre réseau de neurone avec la base de donnée Emnist*
- **Train2.py** *fichier permettant d'entrainer notre réseau de neurones avec les données de Raphael*
- **Word.py** *fichier dans lequel se trouve la classe mot (gère sa création, etc …)*
- **Word-extractor.py** *ce code permettant à partir d'un ensemble de pages, d'extraire chaque mot sous forme de petites images*

Ces fichiers se trouvent dans : **src/neuronal-network-handwritting/src/**

Pour plus de précisions aux niveaux du code, vous pouvez directement aller voir dans les fichiers, ils sont tous commentés.

Je dois impérativement finaliser la première version de mon code qui comporte quelques bug (mais qui fonctionne quand même) avant la semaine prochaine. Nous comptons avec Raphael finir la structure de l'imprimante pour la semaine prochaine et bien avancer la partie électronique.

Important : Ce rapport contient beaucoup de choses non expliqués, mais j'ai passé énormément temps à faire tout ça (plus de 60 heures de travail en tout), mais ce n'est pas toujours facile d'écrire au fur et à mesure son rapport, surtout quand ça ne marche pas (ce qui est arrivé très souvent), mais encore moins facile de s'en remémorer ☹.

Rapport du 14 Janvier

Le week-end, nous avons avec Raphael avancé la structure de l'imprimante. Nous avons donc fabriqué les rails horizontaux et verticaux. Ceci n'a pas été une mince affaire étant donné qu'il fallait parfaitement percer les pièces afin d'éviter tous mouvement nous contrôlé dû à un je.

Nous n'avons pas très bien réussi car nous avions fait quelques erreurs graves au montage précédant. Il nous aura fallu 6h de travail pour faire quelque chose de potable. (A)

On a ensuite pu installer le chariot. (B)

Sur le chariot on a décidé d'installer un bloc qui pouvait recevoir le stylo. (C)

Ce bloc est guidé par 2 tubes afin de pouvoir se déplacer uniquement sur Z. (D)

Pesant environ 60g, nous souhaitions rajouter 2 ressorts sur les tubes afin d'appliquer une force verticale permettant d'appuyer le (stylo + bloc) sur le sol. (E)

Le moteur que nous avions pris afin de lever le stylo à un couple beaucoup trop faible pour pouvoir lever le système (stylo + bloc), on doit donc changer de moteur.

Lundi notre but était d'installer 2 moteurs et la courroie et essayer de faire bouger le chariot.

Nous avons donc fixé 2 moteurs sur les piliers qui soutiennent les rails. (F)

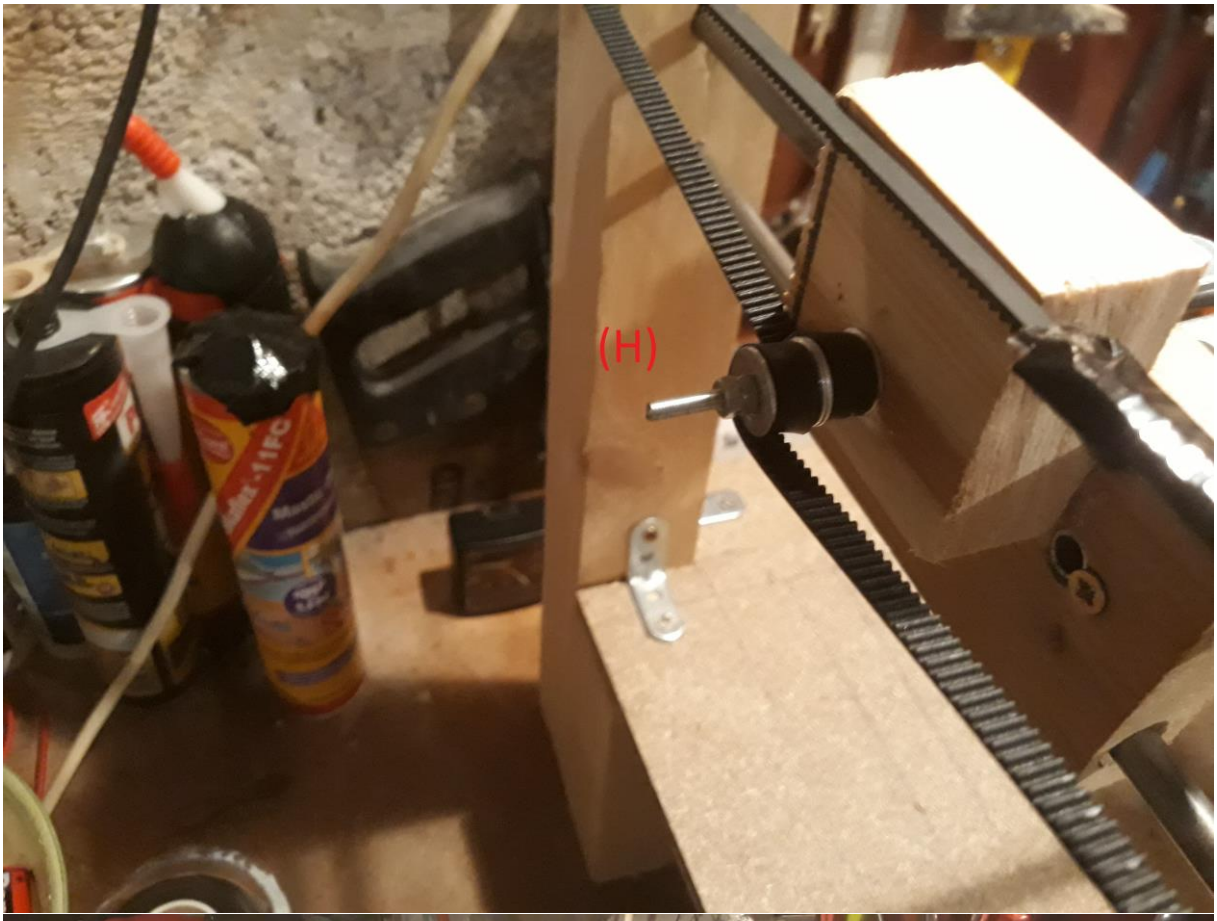
Ensuite nous avons collé d'autres parti en bois afin d'aligner les éléments pour pouvoir installer les poulies. Ceci nous a pris bien 1H. (G)

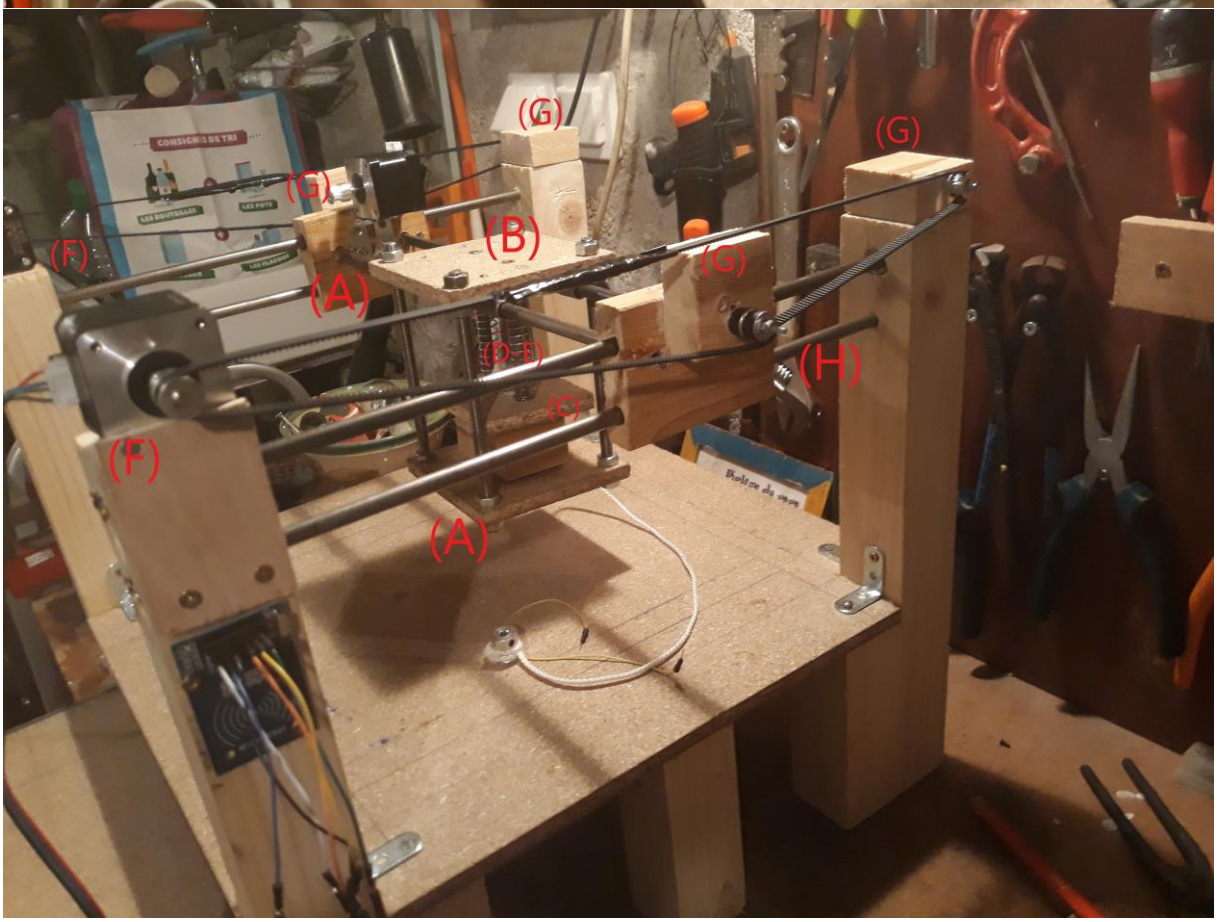
Ensuite nous avons installé des tiges de métal dans le bois afin de fixer les courroies. (H)

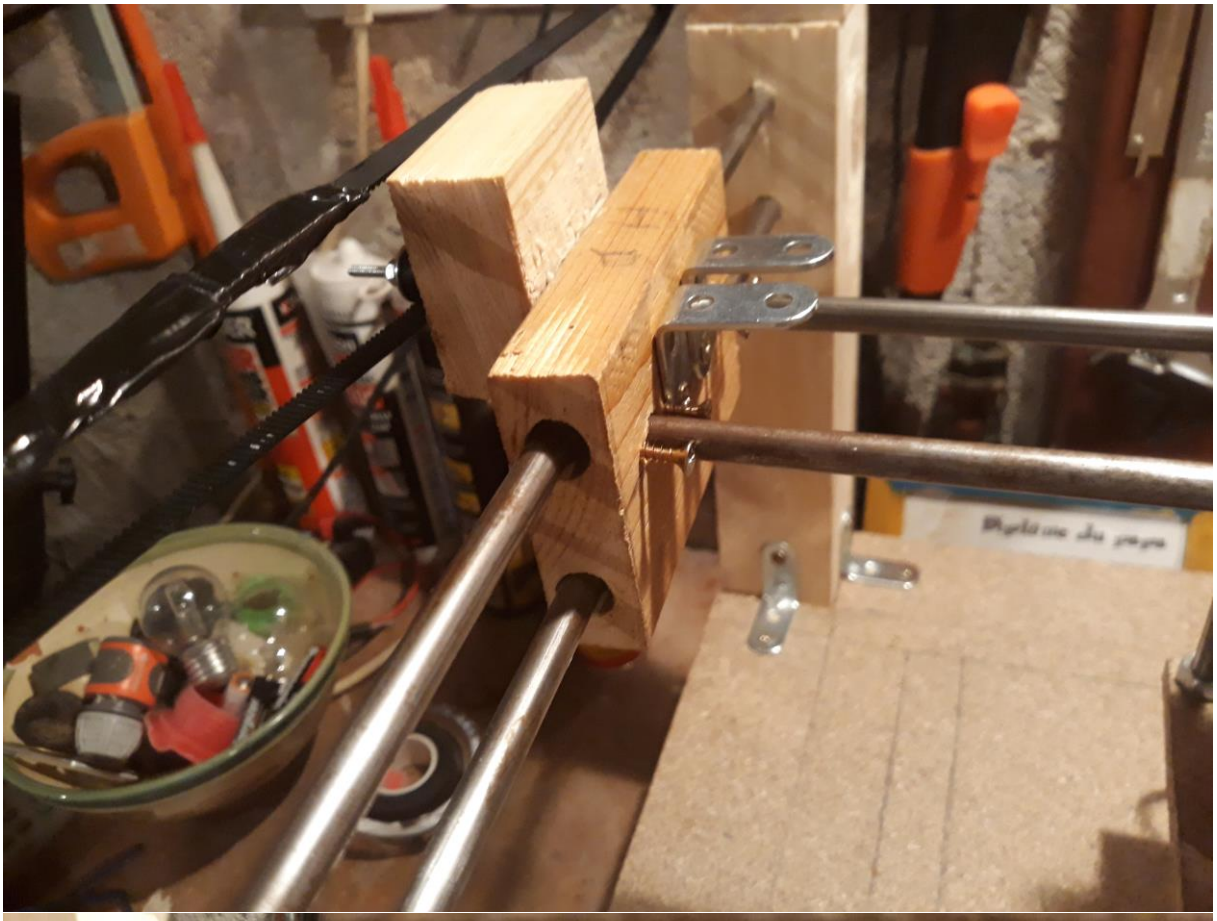
Enfin nous avons retravaillé quelques pièces et nous avons réfléchie sur les futurs travaux à faire au niveau de la structure.

Lundi soir j'ai rajouté quelques éléments comme 2 courroies (I), le module RFID (J), 2 coudes pour la pouvoir mettre un moteur (K).

Photos :


(H)


(G)
(G)
(G)
(F)
(B)
(A)
(G)
(D - E)
(C)
(H)
(F)
(A)



Rapport Antoine

Le projet consistait à faire une imprimante capable d'écrire intelligemment comme une personne normale.

Dans ce rapport je vais essayer de tout résumé et d'expliquer en détail les choses non expliqués.

Explication partie code

Important : Pour des explications plus détaillé de la fonction de chaque scripts python, il faut aller lire les commentaires à l'intérieur de ceux-ci.

1. programmation ordinateur :

Dans les rapports précédents j'ai mentionner le fait qu'il était en fait impossible de faire un réseau de neurone qui allait écrire tous seul, car nous avions des images, et nous avions besoins de séquences.

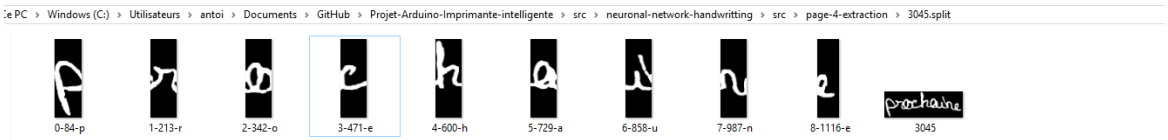
J'ai donc pensé à utiliser un réseau de neurone afin de reconnaitre les lettres dans les mots, de les indexer, et ensuite de pouvoir reprendre, une ou plusieurs lettres adjacentes dans ces mots, afin qu'on est une écriture exactement comme Raphael.

J'ai donc fait un réseau de neurone (fichier train2.py), et j'ai trié des lettres afin que le réseau de neurone s'entrainer a « lire ».

Nota : Pour rappel, j'ai utilisé le code situé dans word-extrator.py qui permet d'extraire des images de lettres ou de mots à partir d'une plus grande image qui peut en regrouper des milliers

Lorsque mes 3000 lettres été triées (*dossier* : src\Python\data\trie-main\trie), j'ai appliqué un algorithme me générant 30 images différentes à partir d'une, ces images sont sur ce GitHub dans Python/data/trie-apres-generation-aléatoire sous forme de .rar

Le code character extrator est le code permettant de splitter l'image d'un mot en lettre afin de pouvoir récupérer ces lettres adjacentes.Des milliers de dossier sont crée 3 exemples :



Dans les noms des images sont inscrit la lettre que l'intelligence artificiel devine. On peut voir tout de même qu'elle estime plutôt bien quand les lettres sont assez espacées.

Sur certains autres mots les résultats peuvent s'avérer minable malheureusement.

Certaines parties de ses mots sont mis dans le dossier src\Python\data\combinaisons afin de les utiliser

Les fichiers word.py, line.py et page.py sont des classes afin de construire une page d'écriture, pour comprendre comment ça marcher, allez lire les commentaires dedans.

J'ai donc fait un code situé dans le fichier generateText qui prend en argument un texte ou un fichier et une port de communication, et qui appelle la classe Page(situé dans page) et qui elle-même appelle Line puis Word afin de forme une image de l'écriture de Raphael :

Voici un exemple :

Texte original :

Comment un individu normalement constitue en arrive a ecrire des choses comme celles que vous allez lire

Un petit historique s impose.Depuis longtemps j ai la manie de glisser des jeux de mots lors des conversations

Les gens de mon entourage ne pouvant plus me supporter je me suis mis a les ecrire

Lorsque je suis entree a l Ecole Polytechnique de Montreal

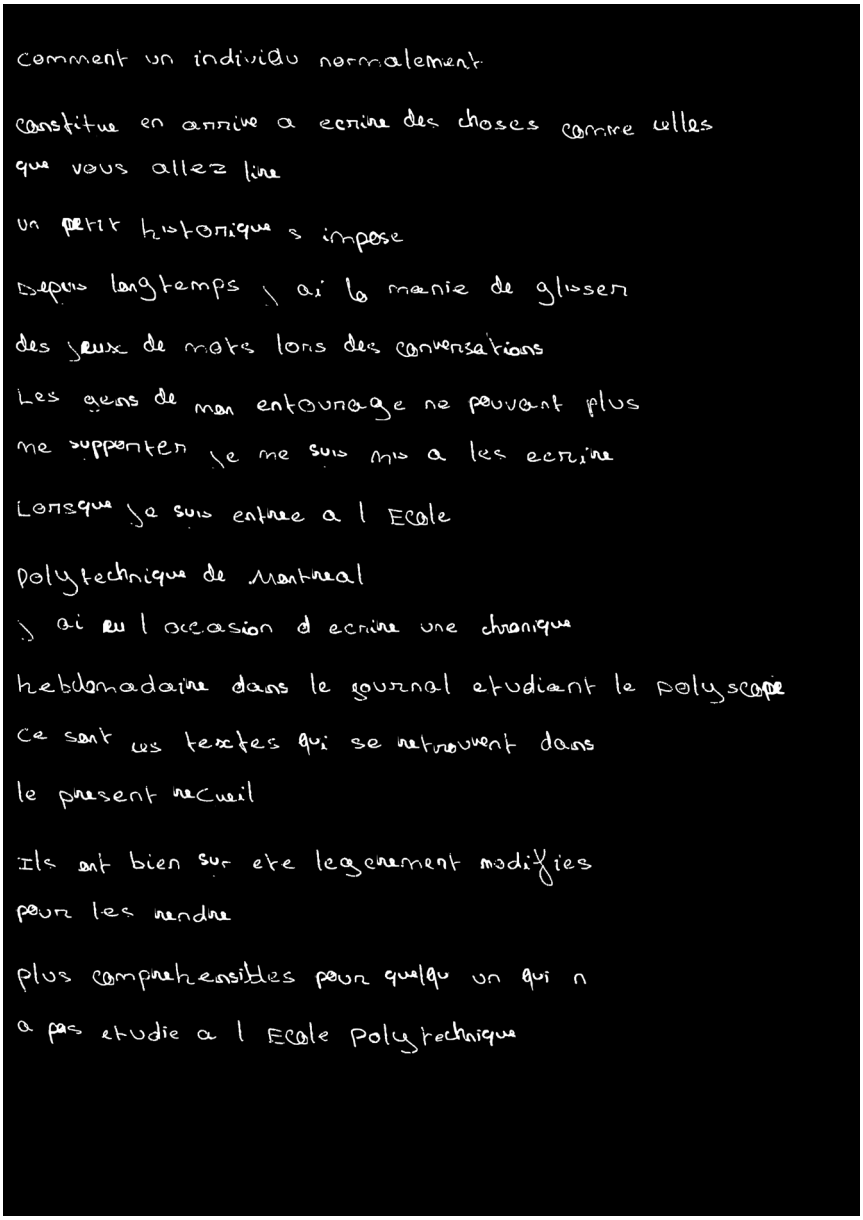
j ai eu l occasion d ecrire une chronique hebdomadaire dans le journal etudiant le Polyscope

Ce sont ces textes qui se retrouvent dans le present recueil.

Ils ont bien sur ete legerement modifies pour les rendre

plus comprehensibles pour quelqu un qui n a pas etudie a l Ecole Polytechnique.

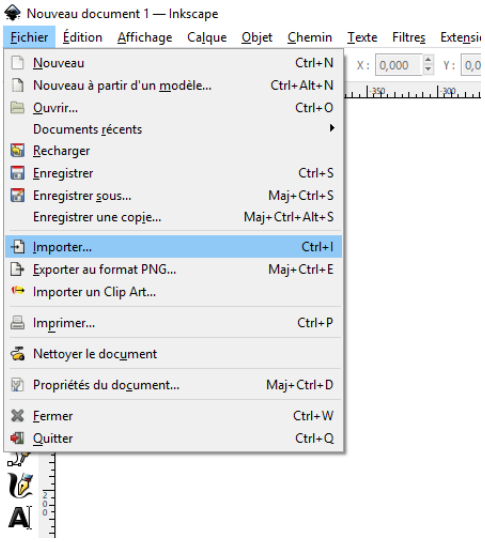
Et voici le résultat :



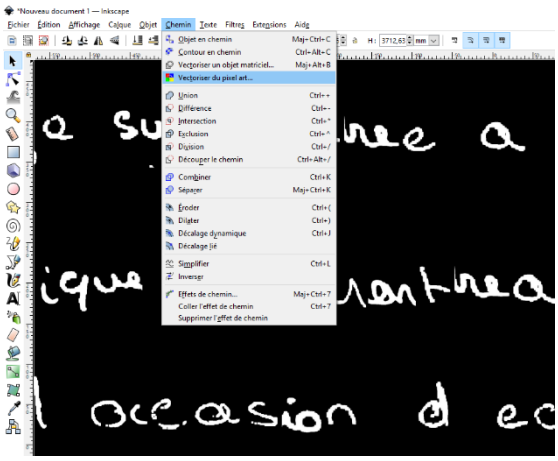
Je suis pleinement satisfait de ce résultat. Tout mon code sur ordinateur fonctionne parfaitement.

Quand j'ai obtenu une image de ce type, j'utilise Inskcape afin de générer un code compris par ma carte et qui permettra de redessiner cette écriture sur papier.

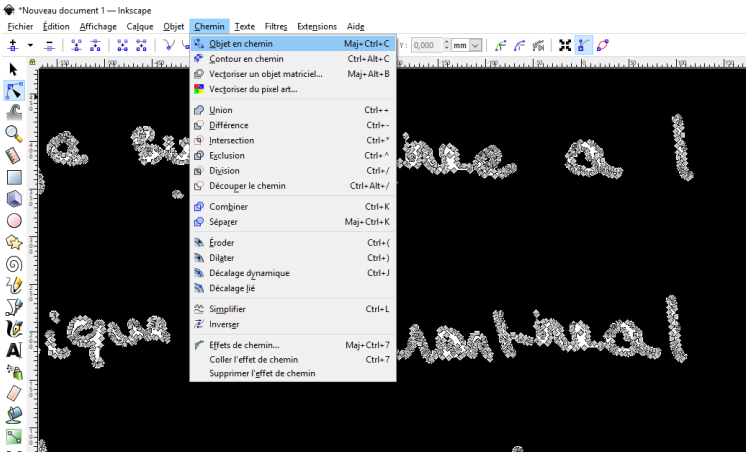
Tout d'abord on importe l'image :



Ensuite on la vectorise :



Après on calcul les chemins :

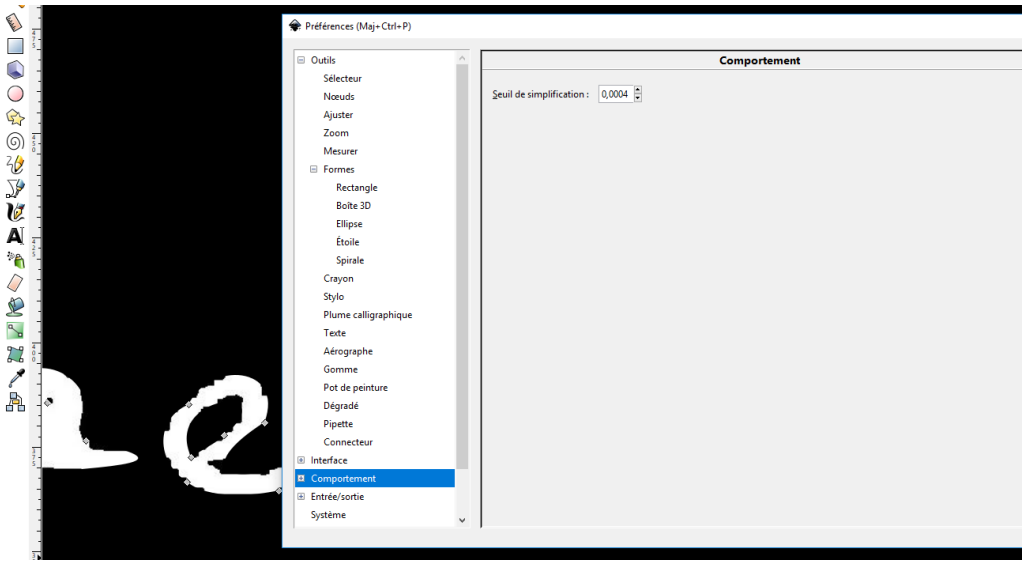


On obient des chemins comme ceci :

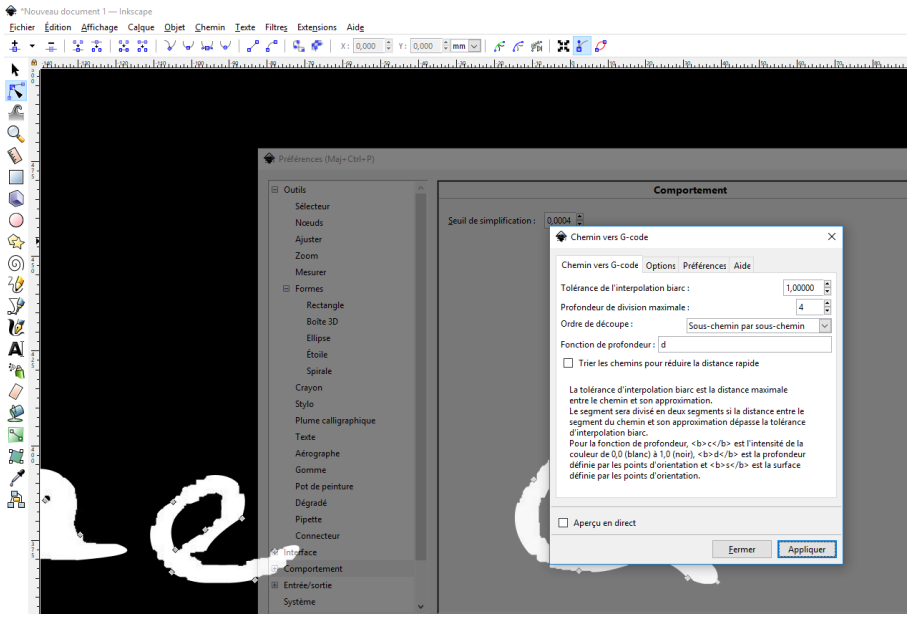
Il est nécessaire de les simplifier, car sinon pour chaque points, l'imprimante va effectuer un tracer, de plus on aura le contour des lettres dessine mais pas la lettre en elle-même.

On s’aperçoit que le nombre de chemins et extrêmement réduit ce qui est parfait pour nous.

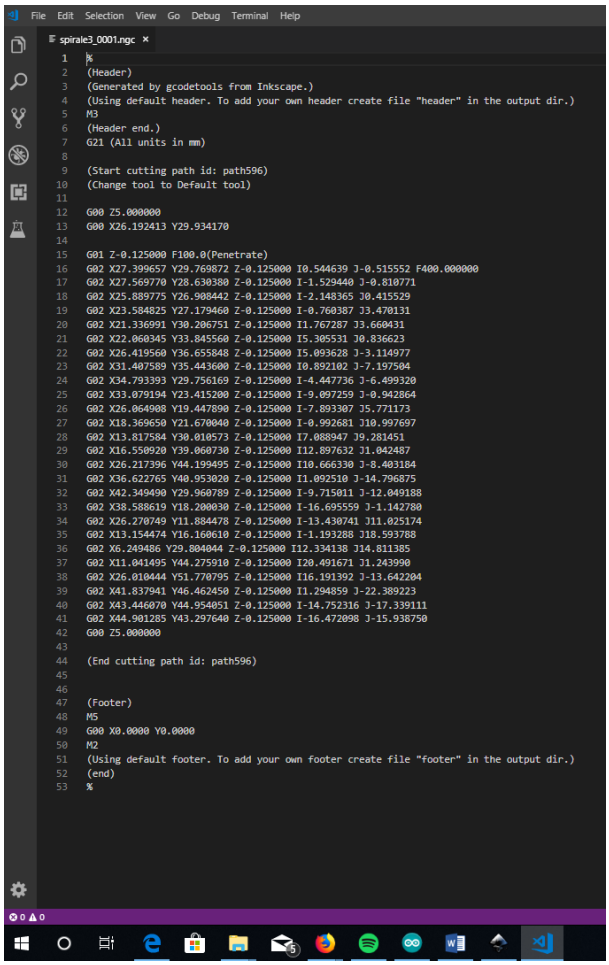
Pour information, on peut choisir le degré de simplification dans les paramètres ici :



Enfin on utilise l’extension gcode de Inskcape (a installer) pour générer le chemin :



Et on obtient un fichier similaire à celui-ci :



Maintenant qu'on a notre code pour l'imprimante, il faut l'envoyer, pour cela il y a le code situé dans le fichier gcodeSender qui s'occupe de tout ça, il lit le fichier ligne par ligne puis l'envoi à la carte arduino via le port serial.

2. Programmation carte arduino :

La carte arduino est programmé afin d'interprété le gcode envoyé par gCodeSender, et de commander les moteurs afin de faire les mouvements décrits par les lignes telles que :

G02 X90.819920 Y202.400407 Z-0.125000 I-3.595403 J8.239973

Je ne vais pas expliquer ici comment cela fonctionne, il vaut mieux plutôt comme déjà dit aller lire les commentaires dans le code.

La carte arduino pilote 4 moteurs, contrôle un module RFID et un écran.

Il a été très difficile d'arriver à ce que la carte arduino interprète le gcode ou les différentes commandes que l'on peut lui envoyer.

Il a fallu faire des fonctions pour calculer le degré de pente, faire un arc de cercle, etc….

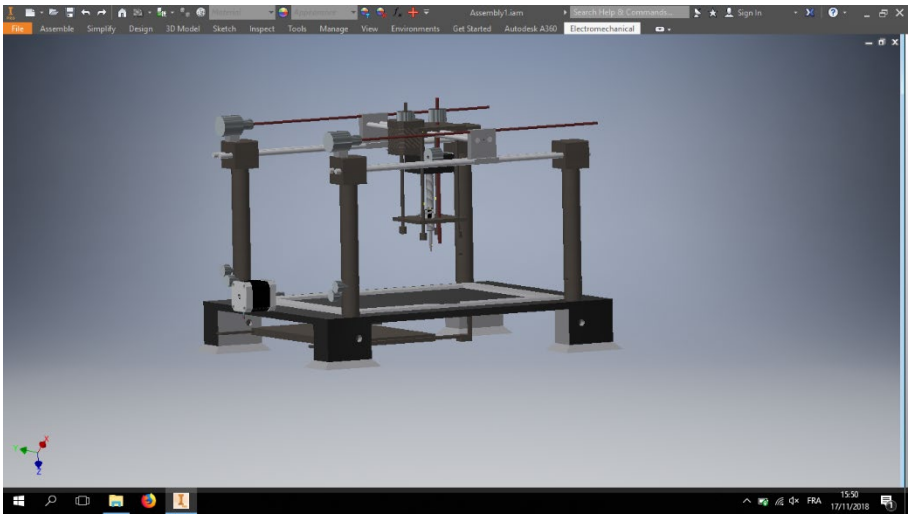
Depuis que j'ai rajouté l'interprétation des commande G02 … et G03 … (arc de cercle), le programme bug un petit peu et nécessite d'être corriger avant tout tentatives d'impressions.

Le code se situe dans scr/Arduino/code princial/printer/

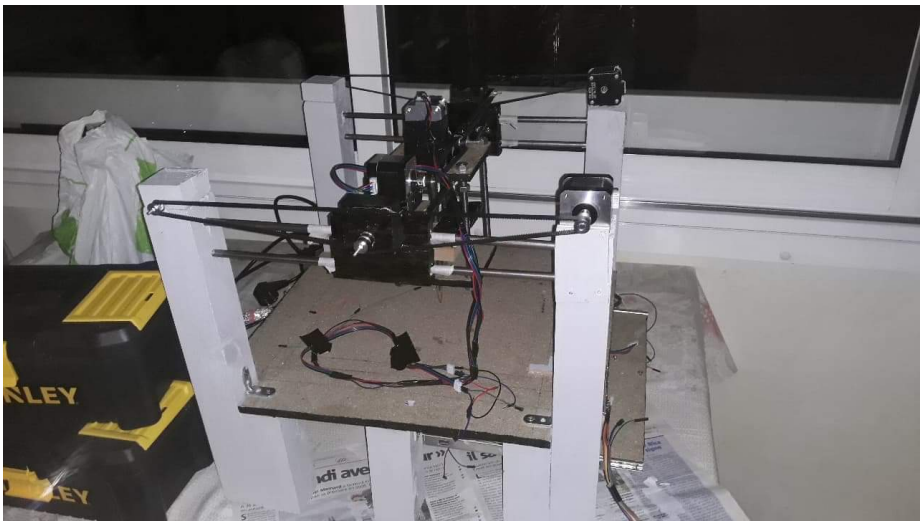
Le module Stepper_custom(situé dans scr/Arduino/librairie/) a été aussi codé afin de faire fonctionner les moteurs

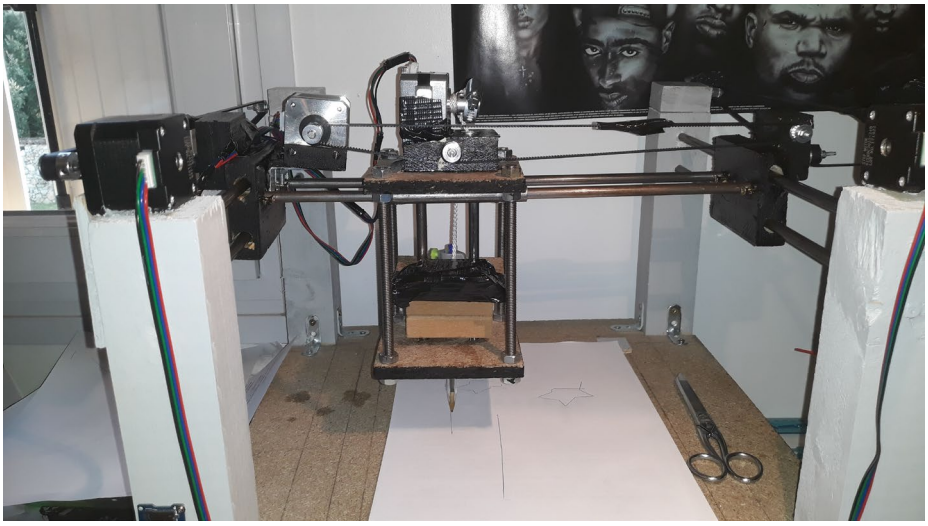
Explication structure et Electronique :

D’abord voici une image de la structure 3d que j’avais imaginé et dessiné :



Et voici des images de la structure finale entièrement construire en bois et métal :

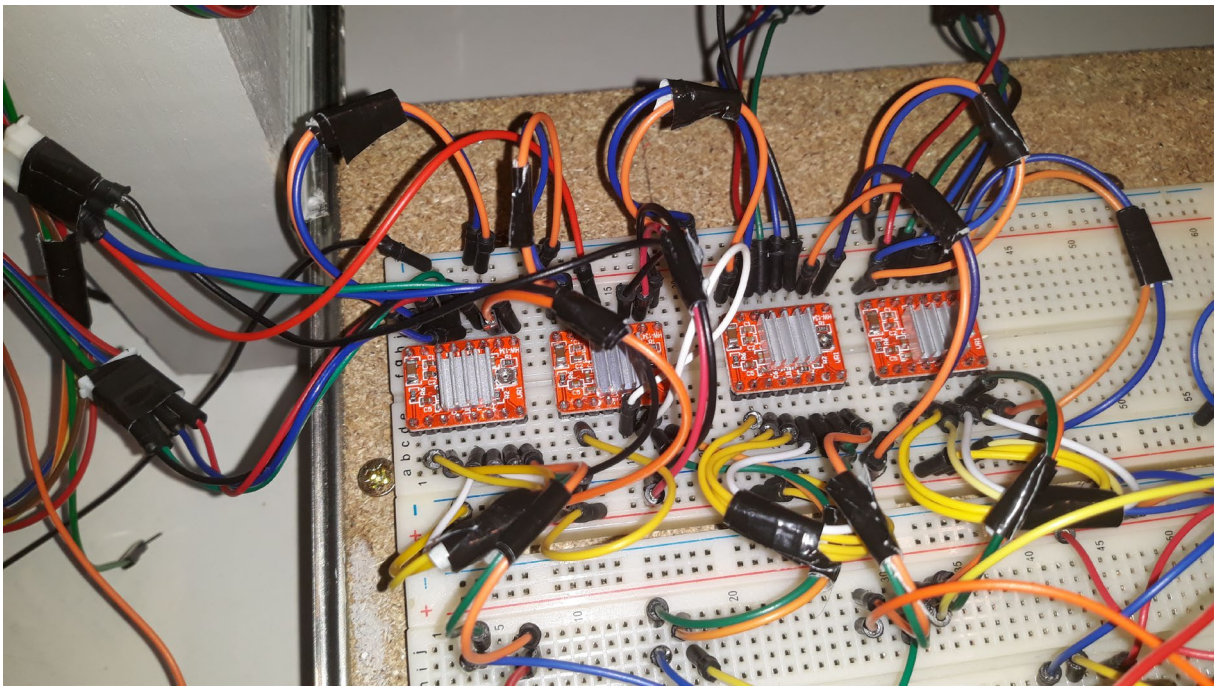




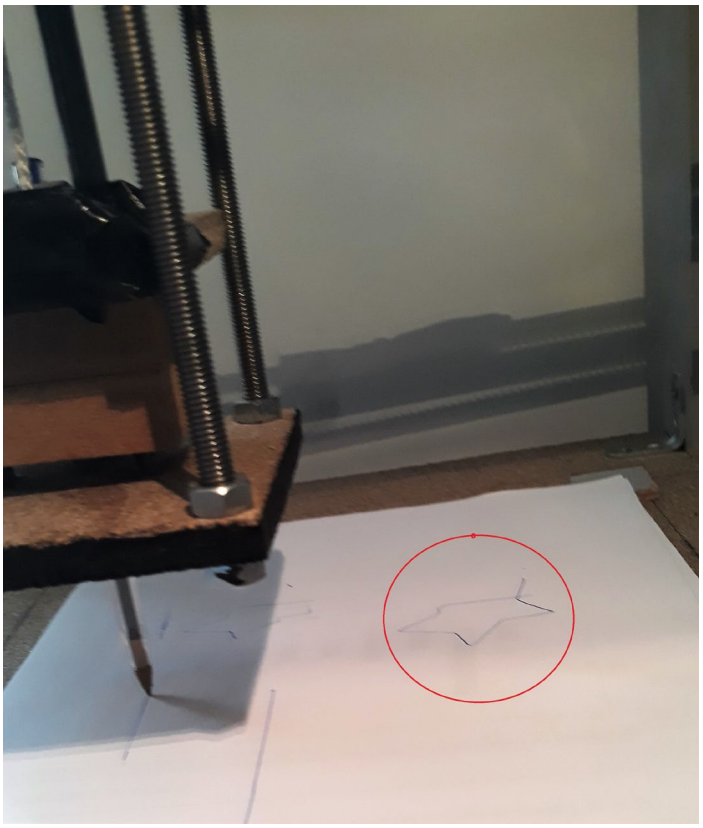
Je trouve que la structure finale est très ressemblante a celle imaginé au tout début

Depuis le dernier rapport ont été rajouté les angles afin de ternir les feuilles, un poids sous le scotch sur le plateau du style, des morceaux de compas pour tenir le style et des légères corrections invisibles sur les photos.

J'ai aussi repeint certaines parties afin de donner un look plus agressif, et bouché les trous fait par erreur.



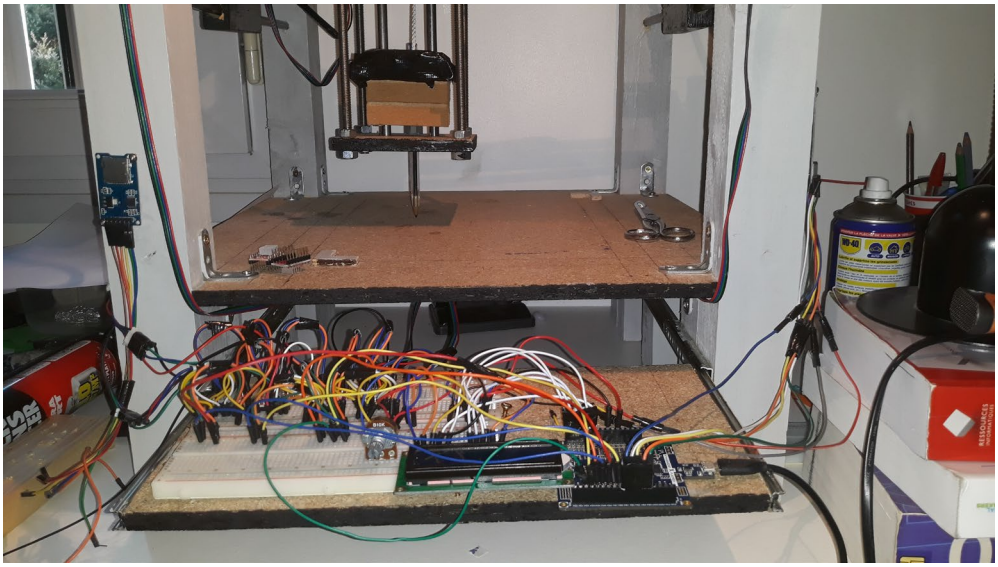
Voici une photo du câblage des drivers qui sera l'échec cuisant dans ce projet. Malheureusement 19 drivers ont été détruits, une petite partie d'entre eux dû à des erreurs de câblage, pour le reste c'est qu'ils sont de très mauvaises qualité et ne supporte pas le travail demandé, même en coupant toutes les minutes ou en refroidissant avec des ventilateurs.



Par exemple on peut voir sur cette image qu'un étoile avait commencé à être dessiner,

Mais au milieu de sa création, le moteur de l'axe X coté droit et tombé en panne, l'étoile n'a pas pu être terminé…

Voici une autre image de l'ensemble des branchements :



On retrouve sur cette photos l'écran, et le RFID sur le côté.

Conclusion :

Ce projet fût extrêmement difficile, et même avec un boulot acharné je n'ai pas réussi à tout faire fonctionner, trop d'imprévu sont arrivés, et je conseil a tout ceux qui voudront faire quelque chose de similaire d'utiliser une imprimante 3d qui interprète nativement le gcode, cela permettra d'éviter la moitié de mes soucis rencontrés.