Raphaël Outhier

LOW LEVEL

Fonctionnement du contrôleur de mouvement

Table des matières

Première partie

Structure du mouvement

1 Définitions

Rappelons d'abord que ce code se destine au contrôle de moteurs pas à pas,et donc aux positions discrètes. En conséquence, toutes les coordonnées seront exprimées dans l'espace des tics moteur. [1]

Pour rappel, la machine possède un nombre fixe (pour la machine, mais pas pour le code) N de moteurs pas à pas, sur lesquels elle devra agir pour faire les bons mouvements.

Soit $\overrightarrow{X} = (x_1, ...x_N)$ le vecteur position de la machine, où x_i représente la coordonnée sur l'axe i, cette coordonnée étant exprimée en tics moteurs. Notons que seules les positions entières peuvent etre exactement atteintes.

L'objectif est de faire suivre à la machine une trajectoire bien définie, dont on connait l'équation. Cette trajectoire peut être représenté par un N-uplet de fonction réelles continues définies sur un intervalle $[a, b]$, $(\Gamma_1, ..., \Gamma_N)$. Pour $x \in [a, b]$, $(\Gamma_1(x), ..., \Gamma_N(x))$ est une position de la machine (aux coordonnées entières ou non !)

Notons que la dimension de x n'est pas nécessairement le temps ! Par exemple :

Une trajectoire linéaire (un trait) de $(0, ..., 0)$ à $(a_1, ..., a_n)$ (le cas $(a_1, ..., a_n) \to (b_1, ..., b_n)$ se ramène au précédent par translation), où il existe $i \in [|1, N|]$ tel que $a_i \neq 0$ peut être représenté par des fonctions $(\Gamma_1, ..., \Gamma_N)$ prenant leurs valeurs dans $[0, a_j]$ avec :

$$x \in [0, a_j] \Rightarrow \Gamma_j(x) = j \text{ et } \Gamma_i(x) = x \cdot \frac{a_i}{a_j} \text{ si i} \neq j$$

Ici les fonctions prennent une distance en entrée.

2 Mouvement

2.1 Mouvement macroscopique

Soit une trajectoire que l'on désire faire suivre à la machine. La première étape est d'extraire une suite $(X_n)_n$ de positions atteignables par la machine(aux coordonnées entières). On va ensuite se rendre successivement à ces positions en utilisant un algorithme de déplacement simple (cf Mouvement Microscopique).

Ces positions successives ne sont pas nécessairement proches, et peuvent être distantes de plusieurs dizaines de tics.

Mentionnons au passage qu'il est impossible de calculer toutes les positions en avance, sous peine de remplir la mémoire du pauvre microcontroleur sur lequel nous ferions tourner le code. Les positions sont donc calculées en temps réel, ce qui consomme évidement du temps.

L'enjeu est de correctement sélectionner la distance entre chaque X_i, en tenant compte du fait que :
- Des positions successives éloignées réduisent le nombre de calcul de position, mais en contrepartie, donnent un mouvement final peu fidèle à la théorique.
- Des positions successives trop proches augmentent le nombre de calcul de positions, mais rendent une trajectoire fidèle à la théorique

1. Pour un axe donné, il est facile de passer de l'espace des tics à l'espace millimétrique, en faisant intervenir le nombre de tics par millimètres, donnée propre à chaque axe. pour un n-uplet d'axes, cependant, c'est plus compliqué.

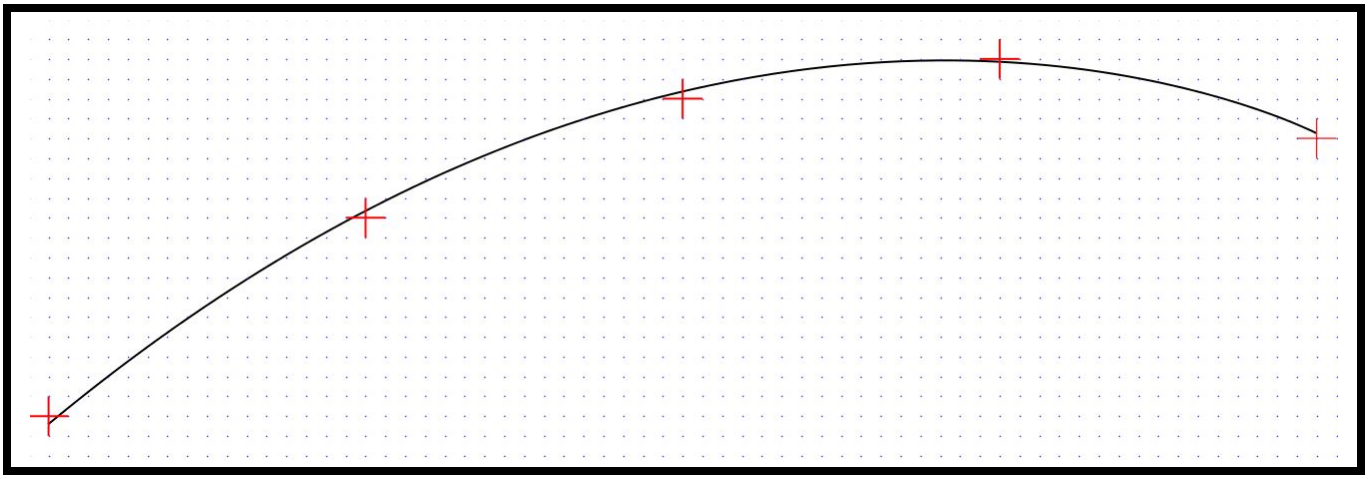

FIGURE 1 – Exemple de discrétisation : trajectoire bidimensionnelle, avec décalage constant de 16 tics

Pourquoi ne pas simplement choisir des positions strictement successives alors (telles qu'entre deux positions successives chaque moteur n'ait pas à bouger plus d'une fois) ?

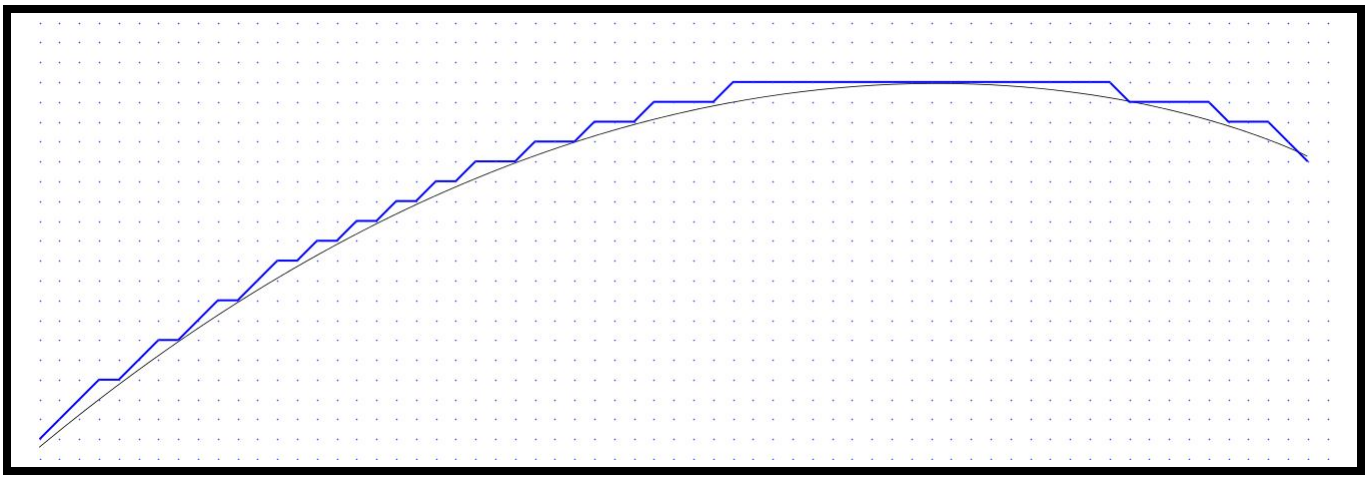

FIGURE 2 – Une discrétisation en positions strictement successives

La réponse est simple et tient en un mot : la vitesse. J'ai dit que ce ne serait pas le thème de ce document, et je m'y tiendrai, mais en deux mots, il faut bien comprendre que temps de calcul d'un vecteur position peut prendre du temps, et que ce temps de calcul est du temps de perdu pour le mouvement : en effet, l'idée générale de la régulation en vitesse, est de bouger les moteurs à des instants donnés, instants séparés par des temps bien déterminés, dont la valeur détermine la vitesse (moins on attend, plus çà va vite).

Avoir des positions strictement successives signifie devoir calculer la position suivante au maximum tous les N tics (N étant le nombre d'axe), ce qui est cause de ralentissement phénoménal.

Ce compromis entre rapidité et précision de la trajectoire fera l'objet d'une section dans le document traitant de la régulation en vitesse.

2.2 Mouvement microscopique

Nous nous intéressons ici au déplacement "réel" de la machine. Nous avons plus haut discrétisé notre trajectoire en positions successives. Le but maintenant est de relier ces points, avec un algorithme simple. On va donc faire prendre une suite de positions à la machine, ces positions étant strictement successives, cad que le nombre de tics moteur sur chaque axe entre deux positions successives est au plus 1 [2]

Supposons que la machine se trouve à la position $(0, ..., 0)$ et doit se rendre à $(x_1, ..., x_N)$

Une idée intuitive, serait de faire x_1 tics sur l'axe 1, puis x_2 tics sur l'axe 2, ainsi de suite jusqu'à l'axe N. Le problème avec cette méthode est (comme toujours d'ailleurs), la vitesse. L'axe 1 va faire des s_1 tics (avec un temps d'attente entre chaque tic) et va ensuite attendre le prochain mouvement pour bouger de nouveau, et il en sera de même pour tous les autres axes. la vitesse sera donc fortement saccadée, ce qui pose de gros problème pour des moteurs pas à pas, qui peuvent rapidement mal bouger et donc fausser le mouvement.

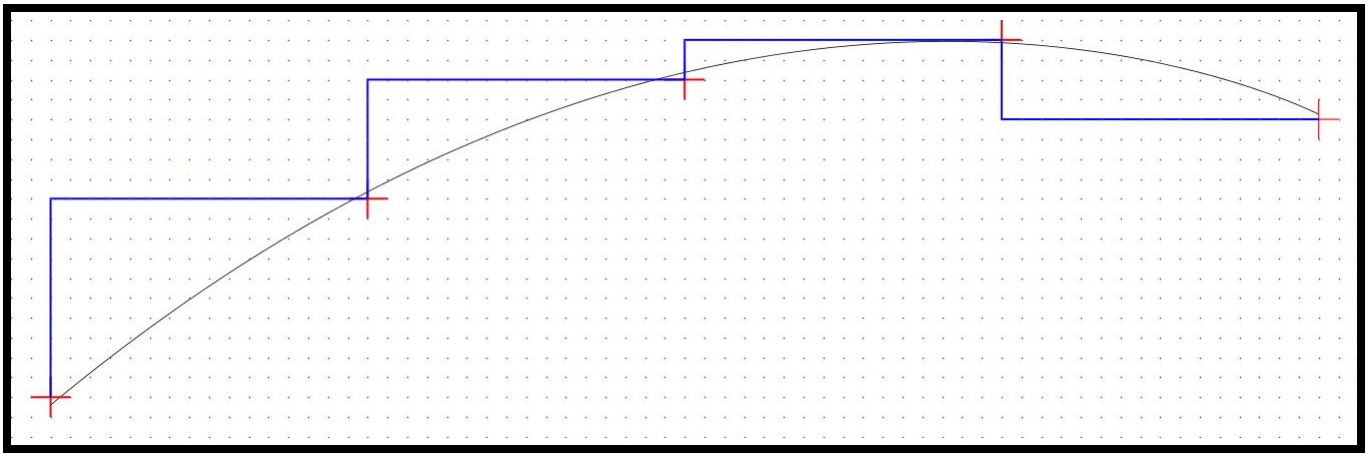

FIGURE 3 – La trajectoire finale (en bleu) est très saccadée

Une idée plus intelligente et donnant des résultats corrects pour le moment est de commencer par bouger tous les axes en meme temps, jusqu'à ce qu'un des axes arrive à sa coordonnée de destination. Ensuite, on bouge tous les axes sauf ce dernier, jusqu'à ce qu'un autre arrive à sa position finale. On bouge ensuite tous les axes sauf ces deux, et ainsi de suite jusqu'à ce que tous les axes arrivent à destination

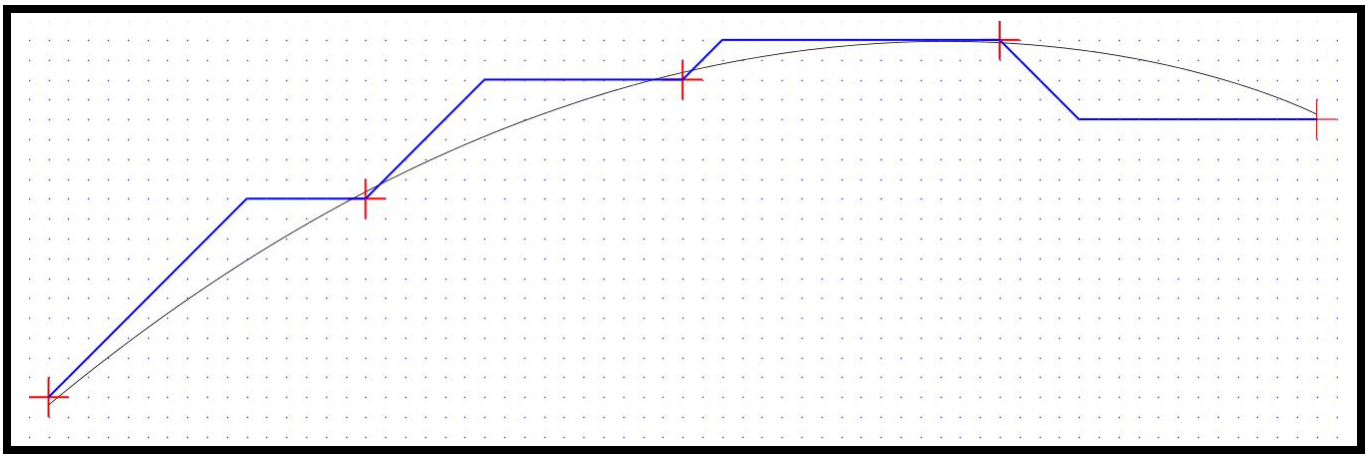

FIGURE 4 – La trajectoire finale est plus lissée

2. X et X' sont strictement successives ssi $||X - X'||_\infty <= 1$

Deuxième partie

Vitesse et accélération

3 Notion de vitesse dans un espace discret

3.1 Direction de déplacement

La partie précédente a traité du mouvement théorique en terme de position. Nous nous intéresserons donc tout naturellement dans cette partie à la vitesse, introduisant une variable (temporelle) supplémentaire au modèle.

Rappelons pour commencer que nous nous trouvons dans un espace discret. Nous avons mentionné dans la partie précédente qu'au cours du mouvement, les positions occupées par la machines étaient strictement successives, soit :

pour X et X' deux positions successives on a $||X - X'||_\infty <= 1$

Étant donné une position actuelle, il existe donc un nombre fini de positions atteignables directement au mouvement suivant : les mouvements simples d'un tic sur chaque axe, et les diagonales.

On reprend l'exemple du mouvement plan, pour bien fixer les idées.

La position actuelle est en noir, les positions accessibles en ne bougeant qu'un axe sont en bleu, et les positions accessibles par diagonale sont en vert.

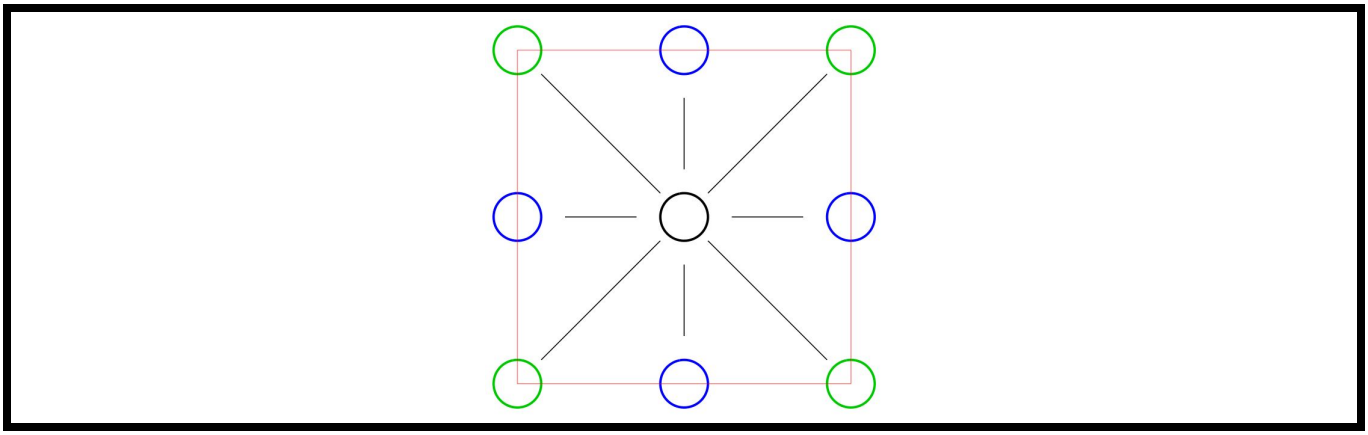

FIGURE 5 – Positions suivantes en 2D

La régulation en vitesse du code a pour but de contrôler, durant le parcours d'une trajectoire donnée, la constance de la vitesse restreinte à un ensemble d'axes, cette vitesse étant calculée par :

$v = ||\overrightarrow{Vr}||_2$ avec V_r le vecteur vitesse restreint aux axes étudié, formule valable si les axes étudiés sont orthogonaux.

Pourquoi réguler la vitesse restreinte ? Un exemple pour mieux comprendre.

Dans une imprimante 3D, si on ne considère que les trois axes du repère cartésiens $\overrightarrow{X}$, $\overrightarrow{Y}$ et $\overrightarrow{Z}$, il est cohérent de calculer la vitesse du chariot comme la norme du vecteur vitesse, défini par $\overrightarrow{V} = (V_x, V_y, V_z)$.

Cependant, Si on se place dans le cas d'une pick-and-place, possédant un chariot 3 axes classique, plus un axe de rotation au niveau du placement du composant, plus un second chariot 2D de sélection des composants, il est absurde de réguler la vitesse des N axes en même temps. On définira une vitesse maximale pour le chariot 3 axes, une vitesse de rotation maximale, et une vitesse maximale du chariot de sélection de composants.

On choisira ensuite laquelle de ces vitesses on souhaite réguler (sa régulation fixant toutes les autres, rappelons que nous accomplissons une trajectoire préalablement définie [3]). On régulera donc non pas la vitesse globale

3. parler de la restriction stationnaire des mouvements

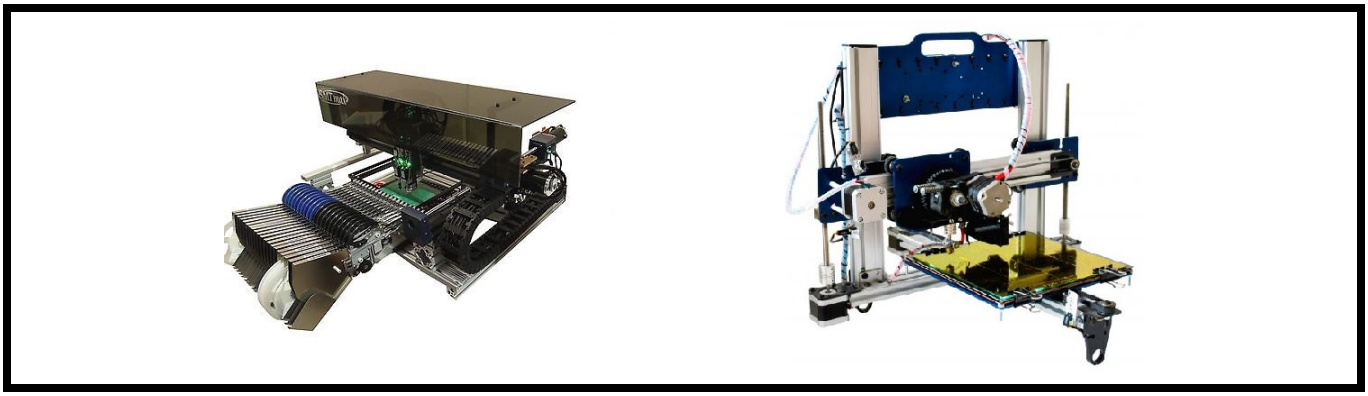

FIGURE 6 – Deux machines : une pick-and-place et une imprimante 3D

mais la vitesse restreinte à certains axes. Notons qu'il est fort probable d'aller plus loin que 3 axes...

Les positions étant strictement successives, étant donné une position actuelle, il existe un nombre limité de positions suivantes possibles, on l'a vu plus haut avec l'exemple du plan : verticale, horizontale, diagonale.

Pour réguler la vitesse, il suffit donc de définir un temps d'attente entre chaque instant, pour chaque direction de déplacement. Venant d'arriver à une position donnée, et connaissant la position suivante, la machine va donc attendre le temps correspondant à la direction à prendre pour atteindre la position suivante, et se rendre à cette position.

3.2 Calcul du temps d'attente

Plaçons nous dans le cas particulier d'un axe, avec les notations suivantes (attention aux unités) :
- T nombre de tics par millimètres de l'axe ($tic \cdot mm^{-1}$).
- V Vitesse de consigne ($mm \cdot s^{-1}$).
- Vt Vitesse de consigne ($tic \cdot s^{-1}$)
- Δ Temps d'attente entre chaque tic ($\mu s \cdot tic^{-1}$)

On a choisi la micro seconde par tic comme unité du temps d'attente car c'est l'unité la plus appropriée compte tenu de la puissance de calcul de l'Atmega. Le temps d'attente minimal actuel sur l'arduino est d'environ $30\mu s$.

Étant donné V, et connaissant T, il faut déterminer Δ.

On a d'abord $Vt = T \cdot V$.

Puis, du fait que $\Delta = \frac{1}{V_t}$ on a $\Delta = \frac{1}{T \cdot V}$

Ce qui donne le temps d'attente lorsque l'on se déplace sur un seul axe.

Pour un déplacement sur M axes simultané (indexés sur $j \in [|1, M|]$), on applique une méthode similaire :

Soient :
- T_j nombre de tics par millimètres de l'axe j($tic \cdot mm^{-1}$).
- V Vitesse de consigne ($mm \cdot s^{-1}$).
- Δ Temps d'attente entre chaque tic ($\mu s \cdot tic^{-1}$)

Le chariot se déplaçant sur une diagonale, il parcourt la distance $d = \sqrt{\sum_j \frac{1}{T_j^2}}$ (mm)

Régulé à la vitesse V,le temps d'attente s'obtient par :

$$\Delta = \frac{10^6 \cdot d}{V} = \frac{10^6 \cdot \sqrt{\sum_j \frac{1}{T_j^2}}}{V} \ (\mu s \cdot tic^{-1})$$

4 Régulation de l'accélération

4.1 Enjeux et difficultés

Pour faire bouger correctement la machine, il va être nécessaire de contrôler son accélération. En effet, passer en un tic à la vitesse de régulation est théoriquement impossible, la vitesse étant l'intégrale de l'accélération et donc continue. En pratique, il est possible (et même nécessaire dans un mouvement aux positions discrètes) de faire varier subitement la vitesse, jusqu'à une certaine limite, les moteurs pas à pas pouvant exercer une force majorée sur le rotor. Si la force n'est pas suffisante pour finir le tic, le moteur reviendra à sa position de départ, introduisant une erreur statique dans le mouvement [4]

Il va être donc nécessaire d'incrémenter graduellement la vitesse, le plus simple étant d'imposer une accélération constante, soit une vitesse linéaire (ou affine dans le cas d'un début de mouvement avec une vitesse non nulle) et donc de calculer l'expression théorique de la vitesse .

Une possibilité calculatoirement simple pourrait être d'exprimer la vitesse en fonction du temps, sachant qu'on débute le mouvement à $t = 0$. Le problème de cette méthode, c'est que le temps, à l'inverse de la position n'est pas une valeur sure pour la machine. Calculer une vitesse en fonction du temps revient à présupposer que l'accélération se fera de manière conforme à la théorique [5], tout retard entrainant un bond brusque de vitesse. Plus grave encore, une décélération correcte suppose la connaissance à priori de l'instant de décélération. Encore une fois, tout retard causera une brusque modification de la vitesse, et (pour l'avoir expérimenté dans les versions initiales du code) une arrivée à destination à vitesse non nulle [6].

On va donc exprimer la vitesse en fonction de la distance parcourue depuis le début du mouvement, variable plus facile à manier pour la machine, et moins dépendante du retard.

Quatre modes de variation de la vitesse sont possibles, augmenter, ou réduire la vitesse, au début ou à la fin du mouvement.

Il est important de comprendre qu'on ne va pas nécessairement revenir à vitesse nulle entre chaque mouvement. En effet, si le haut niveau veut faire un mouvement non prévu par le bas niveau, il va devoir approximer le mouvement dit par une suite de mouvements consécutifs. Plus la discrétisation va être élevée plus la distance parcourue pendant chaque petit mouvement sera petite. S'arrêter entre chaque petit mouvement serait donc une absurdité !

On a (sous certaines réserves), le choix de faire varier la vitesse à la fin du mouvement : supposons que l'on soit, au cours d'un mouvement, à la vitesse V_1, et que le mouvement suivant doive se faire à vitesse V_2, on peut passer de V_1 à V_2 avant de finir le mouvement 1, ou après avoir débuté le mouvement 2. Il est néanmoins nécessaire de pouvoir modifier la vitesse aux deux endroits, car lors d'un mouvement avec arrêt au début et à la fin, on doit accélérer, ET décélérer.

J'ai choisi (arbitrairement) dans le code, que le changement de vitesse se fait au début du mouvement. La machine va donc, connaissant sa vitesse actuelle, accélérer au début du mouvement jusqu'à la vitesse de consigne, qu'elle doive augmenter sa vitesse ou la réduire. Par contre, la modification de la vitesse en fin de mouvement sera uniquement utilisée pour la décélération, ce qui simplifie le code (on ne va jamais augmenter la vitesse en fin de mouvement). Avant de finir cette partie, il est nécessaire d'évoquer un problème douloureux concernant l'accélération, dont la résolution sera ajoutée à ce document d'ici peu (beaucoup de calculs intervenant, elle est un peu longue à rédiger).

Certains mouvements ne permettent pas d'atteindre la vitesse de consigne. Dans la mesure ou l'on a réussi à exprimer la vitesse en fonction de la distance parcourue à accélération et vitesse initiale fixée, on réalise bien que l'accélération d'une vitesse à une autre prend une distance bien déterminée. Par conséquent, si la distance à parcourir pendant le mouvement est inférieure à la distance nécessaire pour passer de la vitesse courante à la vitesse de consigne, la vitesse de consigne ne pourra être atteinte.

Pire encore : si on doit revenir à vitesse nulle à la fin du mouvement, il va falloir compter en plus une distance de décélération. Si les deux distances (atteinte de la vitesse de consigne + décélération) sont plus im-

4. parce que vous vous doutez bien qu'on va pas mettre une codeuse sur chaque moteur pas à pas (? ? ? !) , c'est possible mais pas sur une arduino...

5. ce qui croyez moi est une hypothèse très forte et souvent non vérifiée

6. ce qui est légèrement problématique si on souhaite s'arrêter

portantes que la distance du mouvement, alors on ne pourra pas non plus atteindre la vitesse de consigne.Dans ce cas précis, un enjeu supplémentaire est de déterminer la position de début de décélération (coïncidant avec la position d'arrêt de l'accélération. C'est ce calcul (assez long) qui sera ajouté dans la version suivante de ce document.

4.2 Cas trivial : accélération à vitesse initiale nulle

Notations :
- $\alpha > 0$ accélération$(mm \cdot s^{-2})$.
- l distance parcourue(mm).
- V Vitesse $(mm \cdot s^{-1})$.
- V_c Vitesse de consigne $(mm \cdot s^{-1})$.

Partons de l'équation de la vitesse, en supposant que le mouvement démarre à vitesse nulle [7], et déduisons en l'équation de la distance :

$$V(t) = \alpha \cdot t \ (mm \cdot s^{-1}) \text{ et } l(t) = \frac{\alpha \cdot t^2}{2} \ (mm)$$

On extrait le temps :

$$t(l) = \sqrt{\frac{2 \cdot l}{\alpha}} \ (s)$$

Que l'on injecte dans l'équation de la vitesse :

$$V(l) = \sqrt{2 \cdot l \cdot \alpha} \ (mm \cdot s^{-1})$$

Pour obtenir le temps d'attente

$$\Delta(l) = \frac{1}{\sqrt{2 \cdot l \cdot \alpha}} \ (s \cdot mm^{-1})$$

Le temps d'attente est ici exprimé en $s \cdot mm^{-1}$, ce qui ne nous donne pas accès au temps d'attente réel, ce dernier dépendant de l'axe.

Remarquons qu'on a supprimé (de façon intentionnelle) la vitesse de l'équation. Il va donc falloir arrêter l'accélération au bon moment. On détermine à partir de la formule de la vitesse, la distance d'accélération l_{MAX} vérifiant :

$$V(l_{max}) = \sqrt{2 \cdot l_{max} \cdot \alpha} = V_c$$

$$\text{soit } l_{max} = \frac{V_c^2}{2 \cdot \alpha}$$

4.3 Restriction à un axe

Dans le cas d'un mouvement linéaire, la distance étant parcourue proportionnellement sur tous les axes, on peut se focaliser sur un unique axe, de paramètres α_0 et T_0 respectivement l'accélération et le nombre de tics par millimètres.

$$\Delta(l) = \frac{10^6}{T_0 \cdot \sqrt{2 \cdot l \cdot \alpha_0}} \ (\mu s \cdot tic^{-1})$$

On peut optimiser cette formule pour le calcul informatique :

$$\Delta(l) = \frac{D}{\sqrt{l}} \ (\mu s \cdot tic^{-1})$$

$$\text{avec } D = \frac{10^6}{T_0 \cdot \sqrt{2 \cdot \alpha_0}}$$

7. Une section sera consacrée à la même étude en vitesse initiale non nulle

4.4 Cas plus complet

Les calculs précédents sont faits pour le cas restreint de l'accélération simple, avec vitesse nulle au départ.

On va approfondir un peu les calculs, en étudiant un mouvement accéléré, avec accélération positive ou négative (pour la décélération) avec une vitesse initiale non nulle.

Notations :

- $\alpha \neq 0$ accélération$(mm \cdot s^{-2})$.
- l distance parcourue(mm).
- V Vitesse $(mm \cdot s^{-1})$.
- V^d Vitesse initiale $(mm \cdot s^{-1})$.
- V_c Vitesse de consigne$(mm \cdot s^{-1})$.

Partons encore une fois de l'équation de la vitesse, et déduisons en l'équation de la distance :

$$V(t) = \alpha \cdot t + V^d \ (mm \cdot s^{-1}) \text{ et } l(t) = \frac{\alpha \cdot t^2}{2} + V^d \cdot t \ (mm)$$

On extrait en premier lieu le temps :

$$t = \frac{V - V^d}{\alpha}$$

On doit résoudre l'équation du second degré suivante :

$$\frac{\alpha \cdot t^2}{2} + V^d \cdot t - l = 0$$

dont les deux solutions sont données par :

$$t = \frac{-V^d \pm \sqrt{{V^d}^2 + 2 \cdot \alpha \cdot l}}{\alpha}$$

et, sachant que $t = \dfrac{V - V^d}{\alpha}$, il vient :

$$\frac{V - V^d}{\alpha} = \frac{-V^d \pm \sqrt{{V^d}^2 + 2 \cdot \alpha \cdot l}}{\alpha}$$

soit : $V = \pm\sqrt{{V^d}^2 + 2 \cdot \alpha \cdot l}$

On peut donc maintenant choisir la solution qui nous intéresse sachant que $V > 0$. On en déduit le temps d'attente

$$\Delta(d) = \frac{1}{\sqrt{{V^d}^2 + 2 \cdot \alpha \cdot l}} \ (s \cdot mm^{-1})$$

Optimisons cette formule pour le calcul informatique :

$$\Delta(d) = \frac{D}{\sqrt{l^d + (-1)^s \cdot l}} \ (\mu s \cdot mm^{-1})$$

Avec $D = \dfrac{10^6}{\sqrt{2 \cdot |\alpha|}}$, $l^d = \dfrac{{V^d}^2}{2 \cdot |\alpha|}$, et $s = 0$ si $\alpha > 0$, 1 sinon. Ces paramètres sont invariants au cours du mouvement.

On peut voir l^d comme la distance à la quelle on serait si on avait débuté le mouvement à vitesse nulle.

4.5 Restriction à un axe

Dans le cas d'un mouvement linéaire, la distance étant parcourue proportionnellement sur tous les axes, on peut se focaliser sur un unique axe, de paramètres α_0 et T_0 respectivement l'accélération et le nombre de tics par millimètres.

Notons D_0 le numérateur adapté, et l_0^d la distance fictive adaptée à l'axe concerné, définis par :

$$D_0 = \frac{10^6}{\sqrt{2 \cdot |\alpha_0|}} \text{ et } l_0^d = \frac{{V^d}^2}{2 \cdot |\alpha_0|}$$

On a donc : $\Delta(d) = \dfrac{D_0}{\sqrt{l_0^d \pm l}}$ $(\mu s \cdot mm^{-1})$

Sachant que l'on se place sur un axe particulier, on peut tout à fait se passer du système métrique, en introduisant $T0$:

$$\Delta(d) = \frac{D_0^t}{\sqrt{l_0^t \pm l^t}} \ (\mu s \cdot tic^{-1})$$

avec $D_0^t = \dfrac{10^6}{\sqrt{2 \cdot |\alpha_0| \cdot T_0}}$ et $l_0^t = T_0 \cdot \dfrac{{V^d}^2}{2 \cdot |\alpha_0|}$

respectivement le numérateur corrigé pour le système des tics, et la distance fictive en tics

5 Calcul rapide des temps d'attente par changement d'axe

5.1 Vitesse dans le repère des tics

La formule ci-dessus permet de calculer pour un axe fixé le temps d'attente. Cependant, elle est assez lourde, et ne demande en fait qu'à être utilisée qu'une fois par calcul de vitesse.

Il faudrait théoriquement utiliser cette formule pour chaque axe, car la vitesse en tics par seconde ne va pas être identique sur chaque axe. Cependant, la vitesse en millimètre par seconde, elle, va être la même pour tous les axes (c'est bien le but ! !).

Le repère des tics étant uniquement défini par le nombre de tics par millimètre (que nous appellerons ici T_j pour l'axe j) de chaque axe, on va voir qu'il est possible de retrouver les valeurs des temps d'attente sur chaque axe, et même sur chaque diagonale à partir des T_j.

L'avantage étant bien sur que les T_j, en tant que paramètres mécaniques, ne varient pas au cours du mouvement. On peut donc s'affranchir de calcul en temps réel.

5.2 Changement d'axe

Supposons que l'on ait réussi à calculer le temps d'attente sur l'axe 1.
On souhaite calculer le temps d'attente dans toutes les directions possibles (soit toutes les combinaisons d'axes).

Soit $P \in [|1, N|]$ un groupe d'axes définissant une direction de déplacement pour laquelle on souhaite déterminer le temps d'attente. On rappelle la formule des temps d'attente

$$\Delta_P = \frac{10^6 \cdot \sqrt{\sum_{j\in P} \frac{1}{T_j^2}}}{V}, \Delta_1 = \frac{10^6 \cdot \frac{1}{T1}}{V}$$

soit :

$$\frac{\Delta_P}{\Delta_1} = \frac{\sqrt{\sum_{j\in P} \frac{1}{T_j^2}}}{\frac{1}{T_1}} = \sqrt{\sum_{j\in P} \frac{T_0^2}{T_j^2}}. \text{ On a donc : } \Delta_P = K_{p,i} \cdot \Delta_1 \text{ avec } K_{p,i} = \sqrt{\sum_{j\in P} \frac{T_0^2}{T_j^2}}$$

LOG G BRANLE QUOI C PAS LES MEMES INDICES CORRIGER
Ce qui exprime bien le temps d'attente sur chaque groupe d'axe en fonction uniquement des T_j et de Δ_1.

L'avantage certain des $K_{p,i}$ est qu'ils sont calculables au préalable, et pas pendant le mouvement en lui même.

6 Optimisation des calculs grâce au temps d'attente

6.1 Calculs élémentaires

Comme nous l'avons vu précédemment, la méthode de déplacement nous force à calculer certains paramètres en temps réel au cours du mouvement. On va devoir calculer les positions auxquelles on doit se rendre avec notre algorithme de déplacement, et on va devoir calculer le temps d'attente sur chaque axe que l'on bouge, quand on fera varier la vitesse.

Notons N_p le nombre de calculs intermédiaires nécessaires au calcul de la position, et N_s le nombre de calculs intermédiaires.

Par calcul on entendra un calcul élémentaire dans le langage considéré. En c ce sera une addition, une multiplication, etc... Notons que sur l'Arduino (atmega 8bits) les calculs élémentaires sont faits sur 8 bits soit sur des *char*s, Il va etre important de trouver un algorithme de calcul basé sur la plus petite taille de nombre possibles. On évitera donc les *long*, et on PROSCRIRA les $float$ au plus possible.

6.2 Fenêtres de calculs

Comme on l'a vu précédemment, le parcours d'une trajectoire donnée se résume à choisir une suite de positions proches de la courbe théoriques et espacées de quelques tics entre elles, et de se rendre à ces positions avec un algo simple, à vitesse connue.

Étant à une position, on va devoir calculer la position suivante, calculer les temps d'attente pour la vitesse actuelle (si la vitesse a changé), et se rendre à la position suivante à la vitesse donnée.

Ceci a un gros désavantage : entre le moment ou on arrive à la position actuelle et le moment ou on commence à se déplacer vers la position suivante, on doit calculer la position suivante, et tous les paramètres de vitesse. Cela peut être un calcul long et fastidieux surtout quand on manipule potentiellement des nombres à virgule flottante [8] (cas de la vitesse).

Pour se rendre à cette nouvelle position, on va se déplacer selon notre algo de déplacement microscopique : on va bouger un ensemble de moteurs une fois dans un sens donné, puis attendre le temps correspondant au groupe de moteurs sur lequel on a agi, puis recommencer sur un autre groupe, et ainsi de suite jusqu'à ce que nous soyons à destination.

Cette procédure ne nécessite pas de calcul intermédiaires entre chaque action sur les moteurs, et laisse donc un certain nombre de fenêtres de calcul pour des opérations élémentaires.

On va donc utiliser les fenêtres de calcul disponibles pour calculer en avance, la vitesse et la position que nous devrons atteindre après avoir terminé le déplacement actuel.

Pour cela, il est bien évidement nécessaire de s'assurer que pour tout couple de positions consécutives, l'algorithme de micro déplacement laisse bien $N_s + N_p$ fenêtres de calcul. Si ce n'est pas le cas, la position ou la vitesse ne seront pas calculées, et des aberrations se produiront.

8. Encore une fois, ATTENTION AVEC LES FLOTTANTS, sans FPU (genre comme sur l'arduino, parce qu'on est pauvres...) leur manipulation ruine la performance ! !

Troisième partie

Trajectoires linéaires

Maintenant que nous avons vu tous les points nécessaires pour réaliser correctement une trajectoire quelconque, nous allons nous pencher sur la trajectoire par excellence dans une machine à moteurs, trajectoire linéaire.

J'appellerai par la suite ligne toute trajectoire allant d'une position $A = (a_1, ..., a_N)$ à une position $B = (b_1, ..., b_N)$, vérifiant pour toute position $X = (x - 1, ..., x_N)$ de la trajectoire :

$$(a_i \neq b_i, i \in [|1, N|]) \Rightarrow \forall j \in [|1, N|] \backslash \{i\},\ x_j = a_j + (x_i - a_i) \cdot \frac{b_j - a_j}{b_i - a_i}.$$

Ou de manière moins verbeuse, visualisé dans le repère des moteurs à N dimensions (chaque axe représentant une dimension), la trajectoire est une ligne.

Cette notion de linéarité **dans le repère des tics** est très importante, en effet, les lignes dans le repère cartésien traditionnel (ou avec N axes, chacun ayant la dimension d'une distance) ne sont qu'un cas particulier de trajectoire linéaire avec cette définition.

En effet, si un des moteurs commande une rotation (genre dans une fraiseuse 4 axes) une trajectoire linéaire niveau moteurs ne sera pas du tout une ligne dans notre bon repère cartésien.

Ce type de trajectoire est le plus facile à manipuler et constitue la base d'un système de guidage. En effet, elles sont très simples à réguler en accélération, (calculs plus loin à l'appui), et permettent de reconstituer des trajectoires bien plus complexes, qui seraient impossibles à réguler de manière théorique [9].

Il est donc important pour ne pas dire primordial d'optimiser ces trajectoires.

7 Position

7.1 Classement des axes

Nous supposerons dans toute la suite que le mouvement débute à la position (relative) 0_N, et qu'il termine à la position $D = (d_1, ..., d_N)$, avec $d_i \geqslant 0$, les moteurs pas à pas se commandant de manière similaire dans un sens ou dans l'autre (il faut changer un pin de direction, qui ne change pas l'algorithme.

Commençons par supprimer du mouvement les axes ne bougeant pas. Cela simplifiera grandement les calculs. On va ensuite classer les axes par distance à parcourir décroissante. La raison de ce classement sera donnée par la suite.
On va donc avoir un mouvement de $M < N$ axes, indexés de 1 à M.

Dans la mesure ou on a classé les axes, une notation rigoureuse ferait intervenir une permutation Σ telle que $\Sigma(i)$ soit l'axe à la ième position dans le classement, mais pour ne pas surcharger l'écriture, on considèrera que les axes sont numérotés par leur position dans le classement.

L'axe 1 sera donc l'axe avec le plus de distance à parcourir, et l'axe M celui avec la plus courte distance. Notre variable de référence pour la trajectoire sera la position de l'axe 1.

On a donc $X(x_1) = (x_1, x1 \cdot \frac{d_2}{d_1}, ..., x1 \cdot \frac{d_M}{d_1}) = (x_1, x1 \cdot s_2, ..., x1 \cdot s_N)$ avec $s_j = \frac{d_j}{d_1}$ la pente de j par rapport à 1, pour $1 < j \leq M$.

9. Néanmoins, Low Level permet la régulation partielle en accélération (contrôle de la vitesse au début et à la fin de la trajectoire) de n'importe quelle trajectoire en deux dimension. Ceci fera l'objet d'une partie plus loin dans ce document.

7.2 Optimisation du calcul pour processeur 8 bit

Les pentes sont toutes inférieures à 1, ceci ayant un avantage non négligeable en terme de rapidité de calcul.

Les coordonnées étant entières, les coordonnées seront arrondies à l'entier inférieur pour obtenir des positions atteignables. Sachant que la distance 1 est la plus grande, on peut déterminer une puissance de 2 notée P telle que en multipliant les pentes par P et en arrondissant à l'entier inférieur, en faisant la multiplication entre l'entier obtenu et x_1, et en divisant le résultat par P (division sur entier donc rendant un entier), on ait une erreur inférieure à , et donc qu'on obtienne la valeur théorique arrondie à l'entier inférieur.

En appliquant ce procédé, on a transformé une multiplication sur flottant (je ne le redirai jamais assez, mais ATTENTION AVEC LES FLOTTANTS) en une multiplications sur entier faite une fois pour toute (multiplication de la pente par P), plus une multiplication sur entier à faire à chaque calcul de position et une division par P. Or P est une puissance de 2, et diviser par P revient donc à décaler de $\log_2(P)$ bits vers la droite, une opération implémentée en hardware et donc TRÈS RAPIDE ! de plus $\log_2(P)$ n'a a être calculé une fois, P ne variant pas au cours du mouvement.

On est donc passé d'une multiplication sur flottants (ATTENTION AVEC LES FLOT.. OK j'arrête) à une multiplication sur entier et quelques rotations. Le calcul de position intervenant tout au long du mouvement, c'est un gain de performance colossal !

8 Régulation de l'accélération

8.1 Adaptation de la formule précédente

Rappelons tout d'abord la formule du temps d'attente sur l'axe 1 étant donné T_1 le nombre de tics par millimètre de l'axe 1, une accélération α fixée, pour un départ à vitesse nulle.

$$\Delta_1(d) = \frac{10^6}{T_1 \cdot \sqrt{2 \cdot d \cdot \alpha}} \ (\mu s \cdot tic^{-1})$$

Cette formule fait toujours intervenir le paramètre d, la distance parcourue depuis le début de l'accélération, qui n'est pas accessible immédiatement au cours du mouvement. Cependant, x_1, la distance parcourue par l'axe 1, est immédiatement accessible, car c'est notre variable de référence pour la trajectoire.

On peut donc modifier cette formule, en faisant intervenir α_1 l'accélération perçue par l'axe 1 lors de l'accélération globale :

$$\Delta_1(d) = \frac{10^6}{T_1 \cdot \sqrt{2 \cdot x_1 \cdot \alpha_1}}$$

optimisons donc cette formule pour le calcul informatique :

$$\Delta_1(d) = \frac{D}{\sqrt{x_1}} \text{ avec } D = \frac{10^6}{T_1 \cdot \sqrt{2 \cdot \alpha_1}}$$

8.2 Optimisation

On a donc, pour calculer le temps d'attente sur le premier axe, besoin de deux opérations : un calcul de racine carrée, et une division.

Or, la racine carrée est une opération non implémentée au niveau processeur, lourde en temps de calcul. Il est donc mieux de s'en affranchir si on peut.

Une solution particulièrement efficace, mais réduisant le nombre de pas d'accélération, consiste à ne changer d'accélération que quand x_1 est un carré. x_1 évoluant de manière linéaire (par pas à priori constant et choisi, laissant au moins $N_p + N_s$ fenêtres de calcul), et commençant donc à 0. On initialise à 1 un compteur R donnant la racine carrée entière, et à 1 une variable C donnant le carrée de cette dernière variable.
A chaque fois que x_1 dépasse C, R est incrémenté, et C prend la valeur $R \cdot R$

On peut donc savoir quand x_1 dépasse un carré donné (l'incrément valant rarement 1, il a peu de chances de l'égaler).

Il ne reste qu'une division à effectuer.
Le temps d'attente étant entier, et étant donné qu'il est le résultat de la division de D par un entier, on peut à l'avance le considérer comme un entier et non comme un flottant [10]. On a donc une division sur entiers à réaliser. L'accélération n'allant pas prendre des heures non plus, on peut raisonnablement prévoir quelques cases mémoire pour y placer le résultat de la division de $\frac{D}{i}$ pour quelques i.

Ne reste plus que les changements d'axes à faire, soit des multiplications sur flottants, mais comme faut faire attention avec les flottants [11], mais il doit etre possible de s'en tirer avec des multiplications sur entiers comme pour la position. [12].

8.3 Réduction des calculs

Au début de cette partie, j'ai trié les axes par ordre décroissant selon la distance à parcourir. On va voir maintenant que cela simplifie beaucoup les calculs.

Supposons la machine à une position X, et devant se rendre à une position X'. Elle va s'y rendre en utilisant l'algorithme de déplacement élémentaire.

Posons $(e_1, ..., e_N) = X' - X$. On a, rappelons le, $1 = s_1 \geq ... \geq s_M$

Ceci a pour conséquence que $e_1 \geq ... \geq e_M$. [13]

On se rend compte que si on applique l'algorithme de déplacement élémentaire utilisant les diagonales évoqué en première partie, va commencer par bouger sur la diagonale des axes $1, ..., M$, puis $1, ..., M-1$, puis $1, ..., M-2$, et cetera, jusqu'à bouger uniquement sur l'axe 1. On se limite donc à M directions possibles tout au long du mouvement, ce qui réduit notablement les calculs.

10. Parce que les flottants faut FAIRE GAFFE AVEC!!!!
11. Au moins si j'arrive à vous rentrer cette phrase dans la tête j'aurai pas rédigé ça pour rien!!
12. C'est pas encore implémenté donc j'en parle pas
13. Je n'ai pas fait la démonstration, une preuve ou une infirmation serait la bienvenue, je l'ajouterai si j'ai le temps. En pratique ça marche nickel, chut!

Quatrième partie

Trajectoires 2D quelconques

Le nombre de calculs nécessaires pour tracer une trajectoire quelconque augmente fortement quand N augmente. En effet, en dimension 1, on ne va par exemple, devoir calculer le temps d'attente que pour une dimension. Pour un mouvement plan, il y aura cette fois trois directions possibles. En 3D, il y en aura sept. Plus généralement, il y a $2^N - 1$ directions possibles (on somme les k parmi N pour k allant de 1 à N).

On va donc avoir du mal à trouver un algorithme efficace pour tracer toute trajectoire à N dimension avec N grandissant.

Cependant, le cas du mouvement plan possède deux avantages :

Premièrement, le nombre de calculs intermédiaires pour la vitesse n'est pas énorme, il n'y a que trois calculs à faire, chacun pour une direction. L'algorithme de déplacement élémentaire est donc aussi très simple.
Ensuite, le mouvement plan est très utile et souvent utilisé. Dans les découpeuses laser, les CNC 3 axes, les machines dessinatrices, ou toute machine ayant vocation à tracer des formes planaires, avoir beaucoup de trajectoires implémentés en bas niveau est très utile.

En effet, si une trajectoire n'est pas prévue en bas niveau, il faut, dans le haut niveau, approximer cette trajectoire en une suite de trajectoires acceptées par le bas niveau, souvent des lignes.

Ce procédé a trois inconvénients :

- Premièrement, un découpage d'une trajectoire en petites lignes (par exemple) fait perdre grandement en précision. Si vous avez déjà imprimé des cylindres en 3D, vous aurez certainement remarqué que la face cylindrique est plate par morceaux.

- Ensuite, ce découpage implique la transmission permanente de donnée à la machine (si le gcode est transmis en direct), transmission qui peut comporter des erreurs et nécessite donc une retransmission éventuelle. Cette transmission permanent d'information va perturber le déplacement, et particulièrement la régulation en vitesse, le micro-contrôleur ne pouvant à priori pas faire de multi-threading autre que basé sur interruption, la gestion de la série sur interrupts va devoir avoir lieu pendant le mouvement et donc parasiter la régulation en vitesse.

- Passer d'une trajectoire uniforme à une suite de lignes introduit une quantité considérable de calculs entre chaque ligne. Il faut bien comprendre que la planification d'une trajectoire (même linéaire) est une opération couteuse en terme de calculs et donc en temps. On passe donc d'une seule initialisation à une multitude d'initialisations avant chaque petite ligne ce qui est un gâchis monstre. Sans compter que le maintient de la vitesse entre deux lignes (le non retour à vitesse nulle) nécessite que le temps de planification d'une ligne soit peu élevée par rapport au temps d'un tic moteur (sinon on percevra un arrêt entre chaque ligne). Le temps de planification étant fixé, cela limite FORTEMENT la vitesse.

On voit donc l'intérêt de pouvoir tracer la plus grande variété de trajectoires en bas niveau.

9 Enjeux

9.1 Position

On va avoir à notre disposition les équations des deux coordonnées, que nous appellerons par commodité X et Y, en fonction d'une variable qui cette fois n'est pas nécessairement une distance comme dans le cas simple des trajectoires linéaires.

Pour tracer une ellipse simple de rayons (A, B), par exemple, les equations de la position sont :

$$X(\theta) = A \cdot \cos(\theta)$$
$$Y(\theta) = A \cdot \sin(\theta)$$

où θ est l'angle par rapport au centre de l'ellipse.

Pour une courbe de bezier, la variable de référence sera un réel sans dimension $\alpha \in [0, 1]$.

On notera cette variable α dans toute la suite, prenant ses valeurs dans $[a, b]$

9.2 Conséquences sur la régulation en accélération

Vous vous en rappelez certainement, pour que la régulation en accélération se déroule convenablement, il faut qu'entre deux positions successives l'algorithme de déplacement élémentaire laisse au moins $N_p + N_s$ fenêtres de calcul, avec N_p le nombre de calculs nécessaire au calcul de la position, et N_s le nombre de calculs nécessaire au calcul de la vitesse.

Or, rien ne garantit qu'à pas $\Delta\alpha$ constant au cours du mouvement, le nombre de fenêtres de calculs entre deux positions $(X(\alpha), Y(\alpha))$ et $(X(\alpha + \Delta\alpha), Y(\alpha + \Delta\alpha))$ soit lui constant.

Il faut donc trouver un moyen de s'assurer qu'entre toutes le positions successives, on a bien un nombre suffisant de fenêtres de calculs.

Un problème supplémentaire réside dans l'accélération. En effet, on a réussi à faire disparaitre d (la distance parcourue depuis le début de l'accélération) dans la formule du temps d'attente pour une trajectoire linéaire, mais ce raccourci ne sera pas possible ici. Il va donc falloir maintenir une variable de distance lors de l'accélération, et arriver à déterminer les position de fin d'accélération, et de début de décélération, paramètres à priori inaccessibles de manière théorique [14].

14. En passant, si vous arrivez à me trouver la formule des distances d'accélération dans une ellipse non circulaire contactez moi je suis ultra preneur ! !

10 Résolution pas à pas

Une méthode correcte pour déterminer des paramètres corrects est de simuler le parcours de la trajectoire sans bouger effectivement les moteurs.

10.1 Détermination du pas d'échantillonnage

On va débuter la simulation avec un pas $\Delta\alpha$ arbitraire.

On va ensuite parcourir la trajectoire (sans bouger les moteurs) et ajuster $\Delta\alpha$ pour que entre toutes les positions consécutives on ait au moins $N_p + N_s$ fenêtres. L'idéal étant qu'à un endroit de la trajectoire on ait exactement $N_p + N_s$ fenêtres, car on ne peut plus ensuite réduire significativement le pas, sous peine de passer sous la barrière des $N_p + N_s$ fenêtres. On a donc un $\Delta\alpha$ acceptable.

Il est possible d'améliorer la fidélité de la trajectoire en determinant une partition de $[a, b]$ et en appliquant l'algorithme ci-dessus pour chacun des ensembles de la partition. Lors du parcours de la trajectoire, on choisit $\Delta\alpha$ en fonction de l'ensemble de la partition dans lequel on se trouve.

10.2 Détermination des distances d'accélération - décélération

Le point précédent nécessite d'étudier toute la trajectoire avant de commencer à bouger. On va profiter de cette étape pour déterminer les distances d'accélération.

Pour cela, on va parcourir l'intervalle à partir de a en augmentant α, et à partir de b en réduisant α, ces deux parcours simulés se faisant à la même vitesse, c'est très important. Dans les deux sens, on va donc maintenir deux variables de distances parcourues, et maintenir ces deux distances parcourues le plus proche possible. Il est possible de connaitre la distance d'accélération à partir de la formule de la section 4. Quand on aura dépassé cette distance, on sait qu'on doit arrêter d'accélérer.

Il peut paraitre absurde de faire ces calculs, sachant qu'on peut très bien faire ce contrôle lors de l'accélération réelle. Cependant, la décélération, on l'a vu, nécessite une connaissance à priori du point de début de décélération. En effet, il est tout a fait possible de ne pas pouvoir atteindre la vitesse de consigne, quand la longueur de la trajectoire ou l'accélération maximale sont trop faibles. On va donc accélérer progressivement, et décélérer immédiatement en un point précis. Ce point ne pouvant à priori être déterminé théoriquement, il est nécessaire d'appliquer l'algorithme ci-dessus.

Une fois ces paramètres déterminés, on parcours la trajectoire avec l'algorithme classique, on accélère jusqu'à avoir passé la distance d'accélération, on bouge a vitesse constante jusqu'à arriver au point de décélération, et on décélère jusqu'à arriver à la fin de la courbe.